Razvoj iOS aplikacija

Ševo, Božidar

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Božidar Ševo

RAZVOJ IOS APLIKACIJA

Diplomski rad

Voditelj rada:
dr. sc. Goran Igaly

Zagreb, 2014.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Svima koji su imali bilo kakav utjecaj na moj iOS development...
posebno:
baka Alojzija, mama Lovorka, brat Ante
Sanjin Celeski, Krešimir Prcela
Goran Igaly, Ivica Siladić

Sadržaj

Uvod

Kako sam uopće došao do ovog fakulteta i izrade iOS aplikacija, teme ovog diplomskog rada? U srednjoj školi, negdje do trećeg razreda, planirao sam nakon gimnazije upisati Ekonomski fakultet. Uvijek me zanimala ekonomija, financije i novac. Tada sam saznao da na PMF-u postoji smjer financijska i poslovna matematika i odlučio malo više istražiti o tom studiju. Na kraju sam ipak odlučio upisati preddiplomski studij matematike na PMF-u s planom upisivanja diplomskog studija financijske i poslovne matematike.

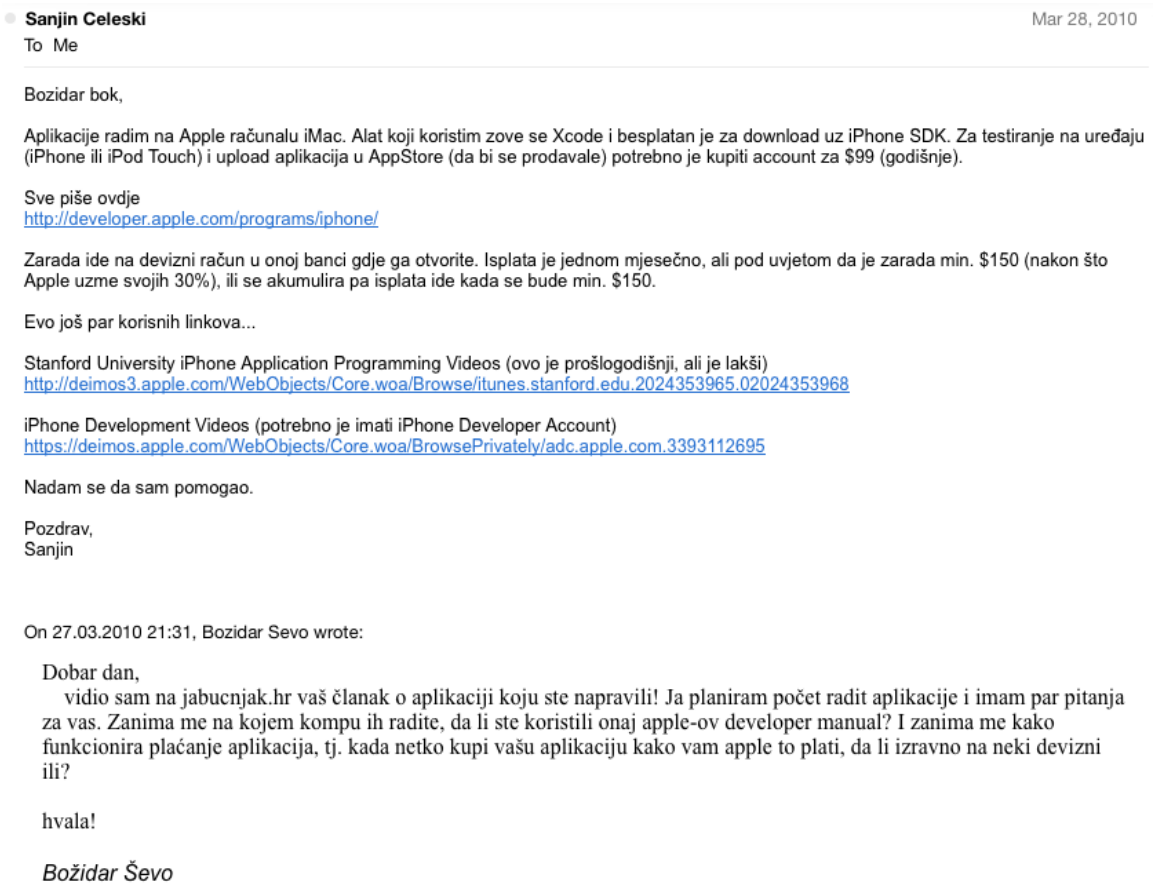
Na prvoj i drugoj godini preddiplomskog studija imali smo nekoliko računarskih, tj. programerskih kolegija kao što su Programiranje 1 i 2, SPA i RP1. Računarski praktikum 1 i profesor Igaly su bili ključni. Na tom kolegiju smo radili C++ i nekako me programiranje počelo sve više i više zanimati. U to vrijeme (8.3.2010.) na fakultetu se održavao otvoreni dan Matematičkog odsjeka[1] i između ostalog na okruglom stolu sudjelovao je Ivica Siladić iz tvrtke Mireo d.o.o. Iskustva g. Siladića su me dodatno inspirirala da krenem aktivnije s programiranjem. Sjećam se da sam nakon tih otvorenih dana cijelu noć razmišljao o dodacima koje bi bilo dobro dodati u GPS software koji radi tvrtka Mireo.

Nekako sam došao do iOS developmenta, tj. tada je bio samo iPhone development jer se operacijski sustav za iPhone zvao iPhone OS, a tek kasnije je nazvan iOS. Koliko me sjećanje služi glavna stavka u kretanju prema iOS developmentu je video koji sam vidio na Youtubeu, a zove se "Apple developer tell their story"[2]. U tom videu prikazane su neke uspješne priče developera koji su imali odlične rezultate s aplikacijama na App Storeu. Nakon toga počeo sam istraživati kako se rade iPhone aplikacije, koji se programski jezik koristi itd. Naišao sam na Sanjina Celeskog iz tvrtke Kreativni odjel iz Rijeke, koji je na jednom forumu vezanom za Apple korisnike pisao o svojoj iPhone aplikaciji. Sanjinu sam 27.3.2010. poslao email s pitanjima kako krenuti s iPhone developmentom, gdje krenuti s učenjem itd., a drugi dan mi je Sanjin odgovorio (slika 0.1) i dao nekoliko odličnih savjeta koji su mi puno pomogli. Da mi nije odgovorio na mail, možda me želja za razvojem iPhone aplikacija ne bi toliko držala, a i da me nije dobro usmjerio možda bih nakon nekog vremena odustao. Jedna od stvari koju mi je savjetovao su predavanja sa Stanforda, točnije s kolegija CS193p - iPhone application development. Ljudi koji vode taj kolegij

[1] http://web.math.pmf.unizg.hr/otvoreni-dan/

[2] https://www.youtube.com/watch?v=qbzSxZ5HkjM

stavljaju video snimke predavanja na iTunes University i bilo tko na svijetu ih može besplatno skinuti i pratiti predavanja kao i studenti na Stanfordu. Zanimljivo je da je iPhone SDK (*Software Development Kit*) izašao 2008. godine, a Stanford je već 2009. imao taj kolegij. Taj kolegij je odlično sastavljen jer pokazuje glavne dijelove i mogućnosti iPhone SDK i osnove Objective-C jezika. Skinuo sam predavanja na računalo i polako počeo učiti iPhone development. Također, u međuvremenu sam prešao na Mac, točnije iMac 21,5", jer se iPhone aplikacije rade na Macu.



Sanjin Celeski Mar 28, 2010
To Me

Bozidar bok,

Aplikacije radim na Apple računalu iMac. Alat koji koristim zove se Xcode i besplatan je za download uz iPhone SDK. Za testiranje na uređaju (iPhone ili iPod Touch) i upload aplikacija u AppStore (da bi se prodavale) potrebno je kupiti account za $99 (godišnje).

Sve piše ovdje
http://developer.apple.com/programs/iphone/

Zarada ide na devizni račun u onoj banci gdje ga otvorite. Isplata je jednom mjesečno, ali pod uvjetom da je zarada min. $150 (nakon što Apple uzme svojih 30%), ili se akumulira pa isplata ide kada se bude min. $150.

Evo još par korisnih linkova...

Stanford University iPhone Application Programming Videos (ovo je prošlogodišnji, ali je lakši)
http://deimos3.apple.com/WebObjects/Core.woa/Browse/itunes.stanford.edu.2024353965.02024353968

iPhone Development Videos (potrebno je imati iPhone Developer Account)
https://deimos.apple.com/WebObjects/Core.woa/BrowsePrivately/adc.apple.com.3393112695

Nadam se da sam pomogao.

Pozdrav,
Sanjin

On 27.03.2010 21:31, Bozidar Sevo wrote:

Dobar dan,
vidio sam na jabucnjak.hr vaš članak o aplikaciji koju ste napravili! Ja planiram počet radit aplikacije i imam par pitanja za vas. Zanima me na kojem kompu ih radite, da li ste koristili onaj apple-ov developer manual? I zanima me kako funkcionira plaćanje aplikacija, tj. kada netko kupi vašu aplikaciju kako vam apple to plati, da li izravno na neki devizni ili?

hvala!

Božidar Ševo

Slika 0.1: odgovor na email od Sanjina Celeskog

Nisam očekivao da će se toliko toga dogoditi i promjeniti i mom životu zbog iOS developmenta, da ću upoznati mnogo novih ljudi i naučiti mnogo novih vještina.

U ovom diplomskom radu objasnit ću neke od glavnih dijelova iOS developmenta i osvrnut ću se na nekoliko vlastitih aplikacija.

Poglavlje 1

Apple

Teško je pisati o iOS developmentu, a da se ne spomene tvrtka Apple[1], koja stoji iza iOS-a i uređaja koje pogoni iOS. Dosta je važno znati povijest te tvrtke, kako su se neki događaji odvijali, koji su ljudi igrali ključnu ulogu i naravno kako je uopće došlo do iPhonea i iOS-a. Također ću navesti neke knjige koje bi bilo dobro pročitati kako bi se dobila bolja slika o kompaniji, proizvodima i ljudima uključenima u Apple.

1.1 Povijest i ključni trenuci

Apple Inc., ili ukratko Apple, je američka međunarodna kompanija sa sjedištem u 1 Infinite Loop, Cupertino, CA 95014. Apple su 1.4.1976. osnovali Steve Jobs, Steve Wozniak i Ronald Wayne. Kasnije je 3.1.1977. Apple inkorporiran kao Apple Computer Inc., a 9.1.2007. su maknuli dio "Computer" i postali samo Apple Inc.

Povijest same Apple kompanije, bez promatranja života osnivača, najbolje je podijeliti u nekoliko dijelova:

- 1976./77. - osnivanje kompanije i prvi proizvod "Apple I"
- 1977. - izlazak "Apple II" računala na tržište
- 1980. - izlazak Apple Inc. na burzu (AAPL[2])
- 1983. - grafičko korisničko sučelje (GUI) s modelom "Lisa"
- 1984. - izlazak računala "Macintosh"
- 1985. - odlazak Jobsa iz kompanije

[1] Apple Inc. - `http://apple.com`
[2] `http://investor.apple.com`

- 1996./97. - Apple kupuje NeXT[3] (tvrtku S. Jobsa) i Jobs se vraća u Apple kao privremeni CEO
- 1998. - izlazak računala "iMac"
- 2001. - Mac OS X, iPod
- 2003. - iTunes Store
- 2007. - iPhone
- 2008. - App Store
- 2010. - iPad
- 2011. - preminuo Steve Jobs
- do danas - Apple pod vodstvom Tima Cooka

1.2 Ključni ljudi

Svaka tvrtka, a pogotovo korporacija kao što je Apple, ima neke ključne ljude koji su ju vodili i/ili danas vode, ljude koji su bili ili su bitni za tu tvrtku. Tako i Apple ima neke ljude bez kojih Apple ne bi bio ovakav danas sigurno. Navest ću samo neke od njih koji su možda u ovom trenutku važniji za spomenuti, a ostale se lako može naći na internetu.

Ronald Wayne

Zanimljivo je da malo ljudi zna uopće da je i on bio osnivač Applea. Radio je s Jobsom u Atariju[4] prije osnivanja Applea. Kad su Wozniak i Jobs osnivali Apple Waynea su "uzeli" kako bi u slučaju nesuglasica između Wozniaka i Jobsa mogao presuditi i pomoći pri donošenju odluka. Pri osnivanju je dobio 10% tvrtke, dok su Wozniak i Jobs svaki imali po 45%. Zaslužan je za prvi logotip tvrtke (Sir Isaac Newton sjedi ispod stabla jabuke), napisao je sporazum o partnerstvu/osnivanju tvrtke i napisao je priručnik za Apple I. Vrlo brzo je prodao svoj udio (za malen novac) koji bi danas puno vrijedio.

[3] `http://en.wikipedia.org/wiki/NeXT`
[4] `http://atari.com`

Steve Wozniak

Poznat kao "Woz" je američki izumitelj, inženjer i programer. Wozniak je sam u potpunosti dizajnirao Apple I i Apple II računala koja su bila prekretnica u revoluciji osobnih računala. Imao sam privilegiju upoznati ovog čovjeka u Londonu na Apps World konferenciji gdje sam izlagao jednu svoju igricu. Kako bi se bolje upoznalo Wozniaka i njegov život svima savjetujem da pročitaju njegovu knjigu "iWoz" [5].

Slika 1.1: Steve Wozniak i ja u Londonu na Apps World konferenciji, 23.10.2013.

Steve Jobs

Poduzetnik, inovator, marketingaš, karizmatik... Razne etikete i epiteti su se davali Jobsu. Dovoljne su samo tri riječi koje pokazuju koliko je velik Jobs bio i koliko je toga napravio:

Apple, NeXT, Pixar[5]. Kako bi se bolje upoznalo Jobsa i njegov život svima savjetujem da pročitaju knjigu Waltera Isaacsona "Steve Jobs" [4]. Steve je nažalost preminuo 2011. godine.

Još neki bitni ljudi

- Mike Markkula
- John Sculley
- Tim Cook
- Jonathan Ive
- Scott Forstall

1.3 Ključni proizvodi

Apple je kompanija poznata po tome da ne radi previše proizvoda, a i od onih koje ima ne radi previše kombinacija ili konfiguracija. Tako su kroz povijest sljedeći proizvodi bili ključni za uspjeh kompanije:

- Apple I
- Apple II
- Lisa
- Macintosh
- iMac
- iPod
- iPhone
- iPad

Jako su zastupljena nagađanja o budućim Apple proizvodima, posebno novim kategorijama proizvoda. Trenutno su popularna nagađanja o mogućem "iWatch" uređaju i mogućem novom "Apple TV" proizvodu.

[5] `http://www.pixar.com`

Poglavlje 2

iPhone i iOS

Teško je iznijeti apsolutnu istinu o tome kako je nastao iPhone i iOS jer je Apple poznat po tajnovitosti. Neke informacije se sigurno nikada neće objaviti u javnosti, ali zato postoje razni intervjui, knjige i ostali izvori, u kojima možemo saznati razne informacije o tome kako je došlo do iPhonea i kako je nastala iPhone revolucija.

2.1 Kako je sve krenulo

U mnogim izvorima pa tako i knjizi "Steve Jobs" [4], otkrivena je jedna zanimljivost. Steve Jobs je prije razvoja iPhonea radio na tabletu. U nekom trenutku je tablet (današnji iPad) projekt stavio na stranu i stavio fokus na razvoj pametnog telefona (iPhone). "Službeni početak" rada na iPhone projektu bio je 2004. kad je Jobs okupio otprilike 1000 zaposlenika, uključujući i Jonathana Ivea, glavnog dizajnera, kako bi radili na super tajnom projektu "Project Purple". Originalni iPhone, prve generacije, Steve Jobs je predstavio 9.1.2007. na MacWorld konvenciji, a u prodaju je pušten 29.6.2007. To predstavljanje je jedno od najboljih predstavljanja bilo kojeg proizvoda u povijesti, a to se slažu i ljudi izvan te industrije.

Operacijski sustav koji je pogonio iPhone u početku se zvao "iPhone OS", a kasnije (2010.) promijenio je ime u "iOS". Kad je iPhone bio predstavljen, na tržištu pametnih telefona vladali su telefoni s fizičkim tipkovnicama, lošim ekranima osjetljivim na dodir itd. Činjenica da je iPhone u biti bio samo jedan veliki ekran osjetljiv na dodir donijela je mnoge koristi. Ekran se mogao u potpunosti prilagoditi ovisno o aplikaciji koja se koristila, tipkovnica bi se pokazala samo kad bi to bilo potrebno, svaka aplikacija je mogla imati jedinstveno sučelje. Prva verzija OS-a nije imala mogućnost da developeri rade nativne aplikacije za uređaj nego su morali raditi web aplikacije koje bi se ponašale slično kao i nativne aplikacije uređaja. Apple je ipak uveo mogućnost da ostali developeri mogu razvijati nativne aplikacije. Apple je 2008. uz iPhone OS 2.0 developerima dao mogućnost

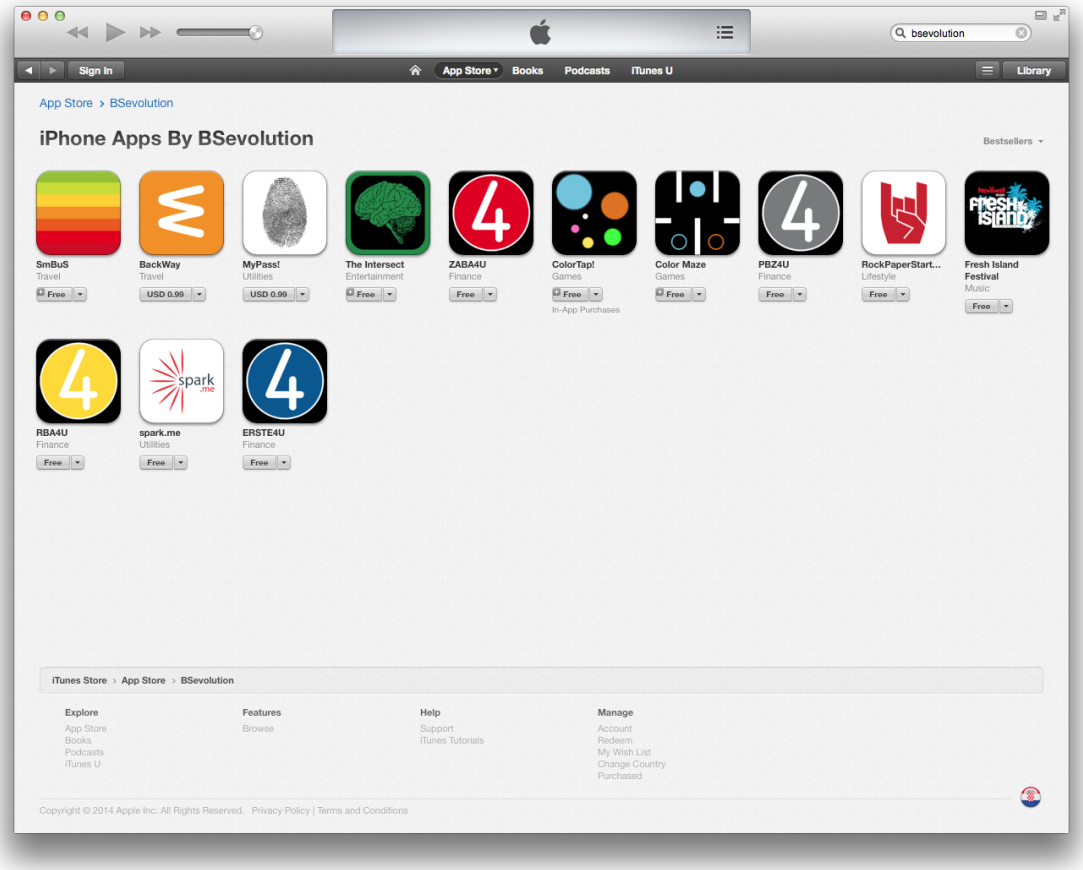
da koriste nativni SDK kako bi razvijali nativne aplikacije za iPhone.

2.2 App Store

"The App Store" je 10.7.2008. pušten u rad. Developerima je omogućeno da na jednom mjestu distribuiraju i prodaju svoje aplikacije. Developer nakon što razvije aplikaciju, pošalje je Appleu na odobravanje i ako/kad je aplikacija odobrena nalazi se na jednom mjestu gdje je svi iPhone korisnici mogu naći. I ono što je najbitnije, korisnici znaju da je svaka aplikacija pregledana od strane Applea i da se sigurno drži zadanih "pravila igre".

Slika 2.1: App Store screenshot, 28.6.2014.

App Store je značio veliku prekretnicu u svijetu razvoja softvera, jer je omogućeno "malim" developerima da uz iste resurse kao i "veliki" rade jednako kvalitetne aplikacije i natječu se na istom tržištu. Također kada se objavljuje aplikacija na App Storeu developer se ne mora brinuti o distribuciji aplikacije, naplati, što itekako olakšava razvoj i poslovanje. Danas (ljeto 2014.) se u App Storeu nalazi više od 1.2 milijuna aplikacija koje su skinute preko 75 milijardi puta. Primjer početne stranice App Storea u iTunes može se vidjeti na slici 2.1. Na slici 2.2 mogu se vidjeti sve moje aplikacije dostupne za skidanje u App Storeu.



Slika 2.2: Moje aplikacije u App Storeu, 28.6.2014.

2.3 iOS verzije

Od prve verzije iPhone OS-a (s puštanjem u prodaju prvog iPhone modela) do danas bilo je nekoliko velikih promjena u iOS verzijama. Svaka nova verzija donijela je mnoge promjene za iOS uređaje, a s dolaskom novih uređaja i njihovim novim funkcionalnostima iOS je također dobivao nove funkcionalnosti. Kao i kod aplikacija tako i kod iOS-a imamo glavne (značajne) nove verzije (2.0) i verzije s manje izmjena (2.0.2).

U svakoj novoj verziji, bilo glavnoj ili manjoj, bilo je dosta novosti, ali ja ću se posvetiti više glavnim verzijama i novostima u njima.

iPhone OS 1

Prva verzija operacijskog sustava za tada jedini iPhone zvala se jednostavno "iPhone OS 1.0". Ta verzija nije imala podršku za izradu nativnih aplikacija od strane ostalih developera (*third-party developers*). Tada je iPhone imao na raspolaganju službene Apple aplikacije koje su već došle s operacijskim sustavom, a ostale aplikacije bi bile u obliku web aplikacija.

iPhone OS 2

Druga značajna verzija iOS-a koja je postala dostupna 11.7.2008. s puštanjem u prodaju iPhone 3G modela. Svi uređaji s 1.x verzijama mogli su nadograditi uređaje na ovu verziju. Najbitniji novitet u ovoj verziji je uvođenje App Storea i mogućnost izrade i objave nativnih aplikacija od strane ostalih developera (*third-party applications*).

Neke od novosti u ovoj verziji:

- App Store
- SDK
- MobileMe
- Novi jezici - norveški, švedski, danski, finski, poljski, nizozemski, korejski, brazilski portugalski
- Kineska, korejska i ruska tipkovnica

iPhone OS 3

Treća značajna verzija iOS-a koja je postala dostupna 17.6.2009. s izlaskom iPhone 3GS modela. U ovoj verziji dodane su funkcionalnosti kao što su cut/copy/paste i MMS. Uređaji s verzijama 2.x mogli su napraviti nadogradnju na ovu verziju.

Neke od novosti u ovoj verziji:

- Push notifikacije
- Cut/Copy/Paste
- Spotlight traženje
- Kompas aplikacija
- Find my iPhone

iOS 4

Četvrta značajna verzija iOS-a, postala je dostupna 21.6.2010. Tada je dosadašnji iPhone OS preimenovan u "iOS". To je prva verzija iOS-a koja nije bila dostupna za sve starije uređaje (zbog performansi itd.). S ovom verzijom objavljen je i iPhone 4. Glavna nova funkcionalnost bila je multitasking.

Neke od novosti u ovoj verziji:

- Multitasking
- Folderi za aplikacije
- HDR fotografiranje
- Facetime
- Game Center
- AirPrint
- AirPlay

iOS 5

Peta značajna verzija iOS-a koja je demonstrirana javnosti 6.6.2011., a postala dostupna 12.10.2011. Prva verzija koja je omogućila nadogradnje bez korištenja iTunes-a (pomoću kabela), tj. *over-the-air updates*.

Neke od novosti u ovoj verziji:

- Siri
- iCloud

- Bežična aktivacija, sinkronizacija i update
- iMessage
- Twitter integracija
- Notification Center

iOS 6

Šesta značajna verzija iOS-a koja je bila najavljena i demonstrirana javnosti 11.6.2012., a postala dostupna u jesen 2012.

Neke od novosti u ovoj verziji:

- Siri na iPadu
- Apple više ne koristi Google Maps nego svoju verziju
- Facebook integracija
- Passbook

iOS 7

Sedma značajna verzija iOS-a najavljena je 10.6.2013., a objavljena je javnosti 18.9.2013. s objavom iPhone 5S i 5C modela.

Neke od novosti u ovoj verziji:

- Flat UI dizajn
- Helvetica Neue Regular postaje sistemski font
- Prozirni dizajn u UI elementima
- Gumbi bez rubova (općenito)
- Back navigacija pomoću geste *swipe*
- Dinamična veličina fontova
- Control Center
- AirDrop
- Foto filtri
- iTunes Radio

Poglavlje 3

Objective-C

3.1 Općenito

Objective-C[1] je objektno orijentiran programski jezik razvijen u osamdesetim godinama prošlog stoljeća. Jezik je nastao pod velikim utjecajem programskih jezika "Smalltalk" i "C", na način da je princip pozivanja funkcija/metoda, tj. slanje poruka u Smalltalku dodan na C.

Objective-C je nadskup C-u i omogućuje objektno orijentirane funkcionalnosti i dinamično izvođenje (*dynamic runtime*). Nasljeđuje sintaksu, primitivne tipove i tok kontrole od C-a te dodaje sintaksu definiranja klasa i metoda.

Upotreba

Objective-C je glavni programski jezik koji Apple koristi za svoja dva operacijska sustava OS X i iOS. Jezik se na OS X i iOS koristi uz adekvatne API-je za te operacijske sustave, "Cocoa"[2] za OS X i "Cocoa touch"[3] za iOS.

Povijest

Objective-C su razvili Brad Cox i Tom Love u osamdesetim godinama prošlog stoljeća u njihovoj kompaniji "StepStone", s fokusom na ponovno korištenje u dizajnu softvera i programiranju. Cox je 1986. u knjizi "Object-Oriented Programming, An Evolutionary Approach" objavio opis jezika Objective-C.

[1] `https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html`

[2] `https://developer.apple.com/technologies/mac/cocoa.html`

[3] `https://developer.apple.com/technologies/ios/cocoa-touch.html`

Tvrtka NeXT 1988. lincencira Objective-C od strane StepStone kompanije, proširuje GCC kompajler za podršku Objective-C i razvija "AppKit" i "Foundation Kit" biblioteke na kojima su bazirani korisničko sučelje i graditelj korisničkog sučelja NeXTStep-a[4]. Apple 1996. kupuje NeXT i koristi Objective-C za daljnji razvoj.

Apple je 2014. na svojoj tradicionalnoj konferenciji WWDC ("Worldwide Developers Conference") predstavio novi programski jezik "Swift"[5], koji se kao i Objective-C može koristiti za Cocoa i Cocoa Touch razvoj. Swift su opisali kao "Objective-C without C", tj. kao "Objective-C bez C-a".

3.2 Osnove programiranja u Objective-C

Pošto je Objective-C nadskup C-a, moguće je bilo koji C program kompajlirati s Objective-C kompajlerom. Također, moguće je koristiti C kod unutar neke Objective-C klase ili općenito nekog Objective-C koda. Objektna sintaksa Objective-C-a proizlazi iz sintakse Smalltalka. Sva sintaksa neobjektno-orijentiranih operacija (primitivne varijable, predprocesiranje, izrazi, deklaracija funkcija, pozivi funkcija) identična je kao i u C-u, a sintaksa objektno-orijentiranih funkcionalnosti proizlazi iz Smalltalkovog stila slanja poruka (*messaging*).

Svakoj osobi koja kreće u iOS programiranje ili općenito želi krenuti učiti Objective-C bilo bi dobro da ima iskustva s C-om i možda nekim objektno orijentiranim programskim jezikom poput C++, Jave itd. Kod kretanja s programiranjem za određenu platformu (npr. iOS) najbolje je polako krenuti s kombinacijom osnova programskog jezika i osnovama samog programiranja za određenu platformu. Tako kod kretanja razvijanja za iOS nije potrebno ići u dubinu samog jezika ili dubinu OS-a i svega što omogućuje, dovoljno je vidjeti osnove i otprilike vidjeti što jezik i OS omogućuju, a onda se kasnije lako može ići u detalje ovisno o tome što čovjeku treba. Tako ću i ja ovdje pokazati neke osnove što se tiče jezika, same iOS platforme i što sve iOS SDK omogućuje i objave na App Storeu. Ima jako puno materije koja bi se mogla obraditi, ali onda bi se nepotrebno išlo u širinu, tako da možda neke dijelove ne pojasnim detaljno ili uopće ne spomenem.

Klase i objekti

Većinu stvari koje ću objašnjavati vezane za Objective-C i programiranje u Objective-C jeziku bit će direktno ili indirektno povezano s razvojem za iOS, tj. uz pomoć Cocoa Touch.

[4] operacijski sustav razvijen od stane kompanije NeXT Computer, kasnije baza za novonastali OS X, a kasnije onda i iOS

[5] `http://developer.apple.com/swift/`

Recimo da se odlučite napraviti neku iOS aplikaciju. Ta aplikacija će biti građena od kolekcije objekata i većinu vremena rada na aplikaciji provest ćete s tim objektima. Ti objekti će biti primjerci Objective-C klasa, neke klase ćete sami kreirati, a neke će biti već napravljene klase koje dolaze s Cocoa i Cocoa Touch koje sam prije spomenuo. Ako kreirate sami svoju klasu, radit ćete slično kao i u ostalim objektno orijentiranim jezicima. Napravit ćete opis klase, sučelje klase koje će sadržavati javna (*public*) svojstva (*properties*) za spremanje bitnih podataka i listu metoda (deklaracije). Deklaracija metode govori koju poruku objekt može primiti i sadrži informaciju o parametrima koji se koriste pri pozivanju metode. Sjetimo se da je u Objective-C poziv metode u biti slanje poruke objektu. Kasnije ćete napraviti implementaciju klase, što znači da ćete napisati konkretan programski kod koji će se izvoditi za svaku metodu napisanu u sučelju klase.

NS

Popriličan broj klasa koje ću spominjati i koje se nalaze u Cocoa i Cocoa Touch imaju prefiks `NS`. Taj prefiks dolazi kao nasljeđe iz povijesti. NeXTStep operacijski sustav tvrtke NeXT koristio je Cocoa koji je počeo kao kolekcija frameworkova. Kad je Apple kupio NeXT, veliki dio NeXTStep OS-a korišten je za OS X pa tako i imena klasa. Cocoa Touch je uveden kao iOS ekvivalent za Cocoa. Neke klase su dostupne i u Cocoa i u Cocoa Touch, a postoji i velik broj jedinstvenih klasa za svaku platformu. Tako da su prefiksi `NS` i `UI` (za elemente korisničkog sučelja u iOS-u) rezervirani za upotrebu od strane Apple-a.

Kategorije

Kategorije su jedna jako zanimljiva i često korisna stvar u programiranju u Objective-C jeziku. Umjesto da za neki zadatak ili funkcionalnost radite sasvim novu klasu, uz pomoć kategorije možete već postojećoj klasi dodati neke nove funkcionalnosti. Kategorije možete koristiti za dodavanje metoda bilo kojoj klasi, imali pristup implementaciji te klase ili ne. Primjerice klasi `NSString` (kao što samo ime kaže, koristi za stringove[6]), možemo dodati neku metodu koja ne dolazi sa samom klasom `NSString` i na taj način možemo koristiti klasu `NSString` bez da radimo svoju cijelu novu klasu samo da bismo dodali tu novu funkcionalnost. Ako imate pristup originalnom kodu klase možete čak i dodati nova svojstva klasi ili mijenjati atribute postojećih svojstava. Više o kategorijama nešto kasnije.

Protokoli

Većina rada na aplikaciji svodi se na to da imamo objekte koji jedni drugima šalju poruke, tj. pozivaju metode. Ponekad je korisno definirati skup metoda koje nisu vezane direktno

[6] pretpostavljam da čitatelj zna što su string, integer, float i ostali primitivni tipovi

za neku specifičnu klasu. Objective-C koristi protokole kako bi definirao grupu vezanih metoda koje mogu biti obavezne ili ne, svaka klasa može "reći" da prisvaja neki protokol, što znači da mora implementirati sve obavezne metode u tom protokolu.

Reprezentacija podataka

Kad se žele reprezentirati neki podaci u Objective-C-u često se koriste Cocoa i Cocoa Touch klase. Tako se primjerice klasa `NSString` koristi za reprezentaciju stringova, klasa `NSNumber` za reprezentaciju raznih tipova brojeva (integer, float, ...). Također može se koristiti primitivne tipove koji postoje u C-u kao što su `int`, `float` ili `char`. Kolekcije podataka su najčešće reprezentirane pomoću neke od kolekcijskih klasa, npr. `NSArray`, `NSSet`, `NSDictionary`, koji se koriste za spremanje drugih Objective-C objekata.

Blokovi (Blocks)

Blokovi su svojstvo koje je dodano u C, Objective-C i C++ kako bi se prezentirao neki oblik jedinice rada. Blokovi sadrže blok koda zajedno sa spremljenim stanjem. Često ih se koristi kod kolekcija, primjerice za enumeraciju, sortiranje, testiranje itd. Također se koriste za planiranje/zakazivanje zadataka koristeći tehnologije kao npr. Grand Central Dispatch (GCD). Više o blokovima nešto kasnije.

Greške (Errors)

Uobičajeno je da se greškama upravlja preko iznimaka (*exceptions*). Objective-C ima sintaksu za reagiranje na iznimke, ali Cocoa i Cocoa Touch iznimke koriste samo za programerske greške, npr. indeks van granica polja, a takve stvari bi trebale biti sređene prije objave aplikacije. Sve ostale greške, npr. nedostatak prostora na disku ili nemogućnost pristupa nekom web servisu, reprezentirane su preko klase `NSError`. Kad se radi aplikacija trebalo bi planirati koje bi se sve greške mogle dogoditi i odlučiti kako reagirati na njih kako bi se pružilo najbolje korisničko iskustvo ako nešto krene krivo. Ako u nekom dijelu aplikacije korisnik treba uploadati neku sliku na web, a nema vezu na internet, trebalo bi na tu grešku reagirati, informirati korisnika da trenutno ne može to napraviti, da provjeri internet konekciju i da pokuša ponovno itd.

Konvencije

Kao kod svakog jezika, tako i kod Objective-C postoje neke konvencije kod pisanja koda. Primjerice imena metoda počinju malim slovom, a ako se ime sastoji od više od jedne riječi onda koristimo *camel case* oblik, npr. `imeMetode` ili `imeMetodeKojaRadiNesto`. Kod imenovanja nije samo bitno korištenje velikih slova, bitno je također da kod bude

što čišći i uredniji kako bi se što lakše čitao i razumijevao. Tako da bi imena trebala što bolje objašnjavati ono što će sama metoda raditi. Ako postoji metodu koja pušta pjesmu nije dobro da se metoda nazove `play`, bilo bi bolje da se nazove `playSongWithName`. U kasnijem poglavlju će se konvencije detaljnije vidjeti kroz ostale frameworkove i primjere.

Detalji

U ovom dijelu ukratko sam iznio neke bitne stvari što se tiče programiranja s Objective-C. U nastavku ću svaku od tih tema nešto detaljnije obraditi.

3.3 Definiranje klasa

Kad pišete neki kod, tj. radite neki softver, većinu vremena radite s objektima. Kao i u ostalim objektno orijentiranim jezicima tako i kod Objective-C objekti služe kako bi se smisleno zapakirali neki podaci i funkcionalnosti. Svaka iOS aplikacija je građena od mreže objekata koji međusobno komuniciraju i rješavaju neke probleme ili zadatke. Neki objekti služe za prikaz korisničkog sučelja, neki za "sirovo" spremanje podataka, dok neki služe za reagiranje na korisnikovo ponašanje (npr. dodir ekrana), neki objekti služe za mrežnu komunikaciju itd. Odlično je to što se kod iOS-a ne mora sve objekte i klase raditi iz temelja nego postoji velik broj biblioteka koje primjerice pruža Cocoa Touch.

Već postoje objekti za spremanje brojeva, stringova itd. Također, postoje objekti za izradu korisničkog sučelja, gumbi (*buttons*), tablice (*table views*) itd. Ti objekti/klase su napravljene tako da ih se može odmah koristiti (*out of the box*), ali ih se može i detaljnije prilagoditi ovisno o potrebama kako bi se dobile potrebne funkcionalnosti i na taj način se ne treba raditi objekte koji će biti zaduženi za te funkcionalnosti.

Na nama je da odlučimo hoćemo li koristiti te biblioteke i objekte te u kojoj mjeri. Nekad će se direktno koristiti bez izmjena, nekad će se koristiti kao bazu za neku novu klasu, a nekad će se nešto iz temelja raditi. Ovisi o developeru koristi li mu određeni objekt iz biblioteka ili ne. U objektno orijentiranom programiranju objekt je instanca, tj. konkretni primjerak neke klase. Slijedi detaljniji opis kako se definiraju klase u Objective-C, kako se deklariraju sučelja itd.

Klase - nacrti za objekte

Kod arhitekture iz jednog nacrta moguće je napraviti više istih kuća, tako i ovdje možemo iz jedne klase napraviti više objekata koji imaju ista svojstva i ponašanja. Tako svaki objekt klase `NSString` ima iste funkcionalnosti bez obzira na to koji niz znakova konkretno reprezentira.

String objekt reprezentira niz znakova, ali ne treba se znati kako konkretno ta klasa sprema podatke. Preko sučelja klase se mi upravlja s objektom i ne mora se znati kako su neke funkcionalnosti interno izvedene. Ako se primjerice koristi objekt tipa skup (*set*), ne zanima nas je li sama realizacija skupa napravljena pomoću liste, polja itd.

Promjenjivo ili ne

Neke klase definiraju objekte koji nisu izmjenjivi (*immutable*), što znači da se sadržaj tog objekta mora postaviti pri kreiranju samog objekta, odnosno ne može se kasnije mijenjati sadržaj. Takvi su primjerice svi `NSString` ili `NSNumber` objekti i ako se želi neki drugi broj naprosto će se morati koristiti drugi objekt tipa `NSNumber`. Neke immutable klase imaju svoje izmjenjive (*mutable*) verzije. Tako za `NSString` postoji `NSMutableString` kojem se može mijenjati niz znakova koliko god se želi. `NSMutableString` ima sve iste funkcionalnosti kao i `NSString`, ali može mijenjati sam sadržaj, tj. znakove u objektu.

Nasljeđivanja

Kad se sjetimo biologije znamo da postoji taksonomija koja životinje svrstava u razne grupe. Te grupe imaju i određene hijerarhije pa tako znamo da su gorile, ljudi i orangutani povezani jer pripadaju istoj porodici (*Hominidae*), ali su različita vrsta.

U svijetu programiranja, objekti su također kategorizirani u hijerarhijske grupe. Ovdje ne postoje vrste, porodice i slično nego su naprosto objekti primjerci neke klase, a klase mogu imati nadklase (roditelj klase) čije funkcionalnosti nasljeđuju. Podklasa od nadklase nasljeđuje sva svojstva i sve funkcionalnosti, a može i dodati neke svoje ili promjeniti (*override*) funkcionalnosti nadklase. Tako je klasa `NSMutableString` podklasa klase `NSString` i ima sve njezine funkcionalnosti, ali dodaje neke metode kao dodavanje, ubacivanje ili brisanje podstringova ili pojedinih znakova.

NSObject

Svim živim organizmima je zajedničko to da su živi. Tako i svi objekti u Objective-C imaju neke zajedničke funkcionalnosti.

Ako se želi da jedan Objective-C objekt radi nešto s drugim Objective-C objektom (ali druge klase) očekuje se da će ta druga klasa imati neke osnovne zajedničke funkcionalnosti. Tako u Objective-C postoji osnovnu (*root*) klasu koju ostale nasljeđuju, a zove se `NSObject`. Ako se radi neku nova klasu potrebno je barem nasljediti `NSObject`, a često se uzima klasa iz Cocoa Touch koja ima neke funkcionalnosti koje su potrebne za novu klasu.

Ako želite definirati neki specifičan gumb u iOS aplikaciji, moguće je da osnovna klasa `UIButton` ne daje sve ono što bi željeli. Tada ima više smisla da nova klasa za specifičan

gumb nasljeđuje `UIButton`, a ne `NSObject`. Jer kada bi nasljedila `NSObject` moralo bi se mnogo stvari dodati kako bi se gumb normalno prikazivao i slično, a sve to već postoji s `UIButton`. A i velika prednost je što će se svi budući popravci (*bug fixes*) ili poboljšanja klase `UIButton` automatski primjeniti i u našoj klasi.

Sama `UIButton` klasa nasljeđuje klasu `UIControl` koja opisuje osnovna ponašanja uobičajena za sve kontrole korisničkog sučelja u iOS-u. `UIControl` nasljeđuje `UIView` klasu koja daje funkcionalosti prikazivanja objekata na ekranu. `UIView` klasa nasljeđuje klasu `UIResponder` koja daje funkcionalnosti reakcije na korisnikov input (tap na ekran, geste ili trešnje). I na kraju `UIResponder` nasljeđuje `NSObject`. Slika 3.1 pokazuje tijek nasljeđivanja klase `UIButton`. Taj lanac nasljeđivanja govori da će svaka podklasa klase `UIButton` nasljediti funkcionalnosti same `UIButton` klase, ali i svih ostalih nadklasa. Što daje klasu čiji objekti se ponašaju kao gumbi, mogu se prikazivati na ekranu, reagiraju na korisnikov input i mogu komunicirati s ostalim Cocoa Touch objektima.

Slika 3.1: nasljeđivanje klase `UIButton`

Sučelje

Klase koje se koriste trebale bi biti dizajnirane tako da skrivaju detalje unutarnje implementacije, a trebale bi samo pokazati kako upravljati s njima i njihovim funkcionalnostima.

Ako se primjerice koristi standardni `UIButton` u nekoj iOS aplikaciji, ne zanima vas kako se točno pikseli prikazuju na ekranu kad se treba pojaviti određeni gumb. Zanima vas jedino kako se može promijeniti svojstva gumba, možda promijeniti boju ili tekst, a da opet budete sigurni da će se prikazati na ekranu kako treba.

Kad se definira neku novu klasu treba razmisliti koje atribute i funkcionalnosti se žele učiniti javnima (*public*). Treba odlučiti kojim će se atributima moći javno pristupiti, možda ih i mijenjati, a treba i odlučiti kako će ostali objekti komunicirati s objektima konkretne klase, dakle treba odrediti i koje će metode/funkcije biti javne. Te informacije idu u sučelje (*interface*) klase koje definira način na koji se planira da će ostali objekti komunicirati s objektima klase. Javno sučelje je odvojeno od "unutarnjeg" ponašanja klase koje čini implementacija klase. U Objective-C sučelje i implementacija su obično u odvojenim datotekama, datoteke sučelja imaju ekstenziju `.h`, a datoteke implementacije `.m`.

Osnovna sintaksa

Ovako izgleda osnovna deklaracija (sučelja) klase:

```
@interface OurNewClass : NSObject

@end
```

Klasa `OurNewClass` nasljeđuje klasu `NSObject`. Javna svojstva i javne metode i ostale funkcionalnosti definirane su unutar `@interface` deklaracije koja završava s `@end`.

Svojstva i vrijednosti

Ako se želi u aplikaciji definirati klasu koja će reprezentirati ljude trebalo bi imati svojstva za stringove koja će reprezentirati čovjekovo ime, prezime, adresu itd. Ta svojstva se deklariraju unutar sučelja na ovakav način:

```
@interface Person : NSObject

@property NSString *firstName;
@property NSString *lastName;

@end
```

Klasa `Person` ima dva javna svojstva koja su primjerci klase `NSString`. Oba svojstva su za Objective-C objekte tako da koriste asterisk (*) kako bi indicirali da su to C pokazivači (*pointers*). Ako se želi osobi dodati godinu rođenja trebalo bi dodati svojstvo tipa `NSNumber`, ali to bi bilo previše jer je za godinu rođenja potreban samo običan cijeli broj pa se mogu koristiti primitivni tipovi iz C-a, u ovom slučaju običan *integer*:

```
@property int yearOfBirth;
```

Svojstvima se mogu dodati i neki atributi (*property attributes*). Sva svojstva objekta klase `Person` mogu se čitati i mijenjati. Pomoću atributa svojstvima se mogu promijeniti mogućnosti, tako se mogu svojstva klase `Person` učiniti *readonly*, tako da će se vrijednosti imena i prezimena osobe moći samo čitati, a neće se moći mijenjati:

```
@interface Person : NSObject

@property (readonly) NSString *firstName;
@property (readonly) NSString *lastName;

@end
```

Deklaracije metoda

Klasi se mogu dodati i neke metode kako bi se obogatile funkcionalnosti klase. U Objective-C objekti komuniciraju međusobno slanjem poruka. Objekt šalje poruku drugom objektu tako da nad tim drugim objektom pozove neku metodu.

Objective-C metode su slične standardnim funkcijama u C-u i ostalim jezicima, ali je sintaksa nešto drukčija. U C-u deklaracija funkcije izgleda ovako:

```
void SomeFunction();
```

Deklaracija njoj ekvivalentne metode u Objective-C izgleda ovako (void je ovdje povratni tip):

```
- (void)someMethod;
```

Znak minus (-) na početku deklaracije metode znači da je ovo *instance* metoda, tj. da se ona može pozivati na primjercima klase, dok *class* metode imaju znak plus (+) i one se mogu zvati na samoj klasi.

Metode naravno mogu primati i argumente, jedan ili više njih. Ovdje se sintaksa dosta razlikuje od sintakse u C-u, posebno kad u metodi ima više argumenata.

```
// C:
void SomeFunction(SomeType value);

// Objective-C:
- (void)someMethodWithValue:(SomeType)value;
```

Vidi se da sama deklaracija metode sliči na priču u kojoj pričamo što metoda radi s kojim argumentima, što se odlično vidi kod metoda s više argumenata i daje poseban čar (barem meni) jer su deklaracije metoda opisne i jasno se vidi što se događa i Objective-C čini jako čitljivim jezikom:

```
- (void)someMethodWithFirstValue:(SomeType)value1 secondValue:(
    AnotherType)value2;
```

Zanimljiva stvar kod deklaracije metoda jest da se kod implementacije te metode ne trebaju koristiti ista imena vrijednosti/argumenata, jedino je bitno da se potpisi (signatures) poklapaju. Što znači da isti trebaju biti ime metode, povratni tip i tipovi argumenata.

Ova metoda ima isti potpis kao metoda prije:

```
- (void)someMethodWithFirstValue:(SomeType)info1 secondValue:(
    AnotherType)info2;
```

Ali ove dvije metode imaju različit potpis od metode prije:

```
- (void)someMethodWithFirstValue:(SomeType)info1 anotherValue:(
    AnotherType)info2;
- (void)someMethodWithFirstValue:(SomeType)info1 secondValue:(
    YetAnotherType)info2;
```

Imena klasa

U aplikaciji koja se radi sve klase moraju imati jedinstvena imena, bile to klase koje smo mi sami radili ili klase koje se već nalaze u nekim bibliotekama. Ako se kojim slučajem u projektu napravi klasa istog imena neke druge klase koja se koristi u projektu, dobit će se grešku u kompajliranju. Tako da je uobičajeno stavljati prefikse klasama. Ako se aplikacija zove "BackWay" i radimo neku klasu koju bi se koristilo za spremanje podataka o lokaciji korisnika ne bi bilo loše nazvati ju `BWLocation`. Prefiksi uobičajeno imaju 2, 3 ili više slova. Na taj način klasa se povezuje s projektom ili s nama samima ako koristimo inicijale kao prefiks. Imena metoda i svojstava neke klase moraju biti jedinstvena samo za konkretnu klasu.

Implementacija klase

Kad se definira sučelje klase, uključujući svojstva i metode koje će biti javne, trebalo bi napisati i kod implementacije klase. Kao što sam rekao prije sučelje klase se uobičajeno piše u odvojenu "header" datoteku, koja ima ekstenziju `.h`, a implementaciju Objective-C klase pišemo u datoteku s ekstenzijom `.m`. Kad su tako odvojeni sučelje i implementaciju potrebno je reći kompajleru da prvo učita datoteku sučelja prije nego pokuša kompajlirati kod implementacije. Objective-C ima predprocesorsku naredbu `#import` (ne završava s ";" kao što je slučaj u C-u), koja je slična naredbi `#include` u C-u, ali se čak i brine da je datoteka samo jednom uključena pri kompajliranju.

Sintaksa implementacije

Ako se u sučelju klase definiraju neke metode trebalo bi ih i implementirati.

```
// XYZPerson.h datoteka
@interface XYZPerson : NSObject
- (void)sayHello;
@end
```

Implementacija:

```
// XYZPerson.m datoteka
#import "XYZPerson.h"
```

```
@implementation XYZPerson
- (void)sayHello
{
  NSLog(@"Hello, World!");
}
@end
```

Ovdje se koristi Objective-C string literal `@"Hello, World!"`. Stringovi su jedan od tipova klasa u Objective-C koje imaju mogućnost ovakvog stvaranja objekata. Ovdje se implementacija sastoji samo od poziva funkcije `NSLog()` koja ispisuje poruku na konzolu. Ta funkcija je slična standardnoj C funkciji `printf()`, a također i prima različit broj argumenata, od kojih prvi mora biti Objective-C string.

Još jednom da napomenem da je konvencija da imena metoda počinju s malim slovom, a nastavljaju s *camel case*. I jako je važno da se imena metoda pišu desktriptivno kako bi se lako čitao kod. Apple-ovo integrirano razvojno okruženje (*Integrated Development Environment* - IDE) za razvoj iOS i OS X aplikacija "Xcode"[7] će pri pisanju koda samo uvlačiti kod i raditi slične automatizirane stvari koje se lako mogu i prilagodit u postavkama.

I klase su objekti

U Objective-C i sama klasa je objekt, objekt tipa `Class`. Klase ne mogu imati svojstva definirana koristeći sintaksu koju sam pokazao prije, ali mogu primati poruke, tj. imati metode.

Tipično korištenje klasne metode su *factory methods*, koje su alternativa za pozivanje procedura za alokaciju i inicijalizaciju neke instance klase. Slijede primjeri takvih metoda za klasu `NSString`:

```
+ (id)string;
+ (id)stringWithString:(NSString *)aString;
+ (id)stringWithFormat:(NSString *)format, ...;
+ (id)stringWithContentsOfFile:(NSString *)path encoding:(
      NSStringEncoding)enc
  error:(NSError **)error;
+ (id)stringWithCString:(const char *)cString encoding:(
      NSStringEncoding)enc;
```

Kao što sam već rekao klasne metode imaju znak (+) što ih razlikuje od *instance* metoda koje imaju (-). Klasne metode se također deklariraju u sučelju klase i implementiraju unutar `@implementation` bloka za konkretnu klasu.

[7] `http://developer.apple.com/xcode/`

3.4 Rad s objektima

U prošlom dijelu pokazao sam osnove sintakse te pisanja sučelja i implementacije klase, dok ću u ovom dijelu pokazati kako se šalju poruke objektu itd. Prije nego se objekt može koristiti treba ga pravovaljano stvoriti koristeći kombinaciju alokacije memorije za njegova svojstva i potrebne inicijalizacije unutrašnjih vrijednosti. Ovaj dio će pokazati detaljnije kako se pozivaju metode, kako ih ugnježdavati, kako stvoriti objekte i provjeriti da li su stvoreni bez grešaka pri inicijalizaciji ili alokaciji.

Slanje i primanje poruka između objekata

Postoji nekoliko načina kako slati poruke između objekata u Objective-C, ali najčešći način je koristeći uglate zagrade:

```
[someObject doSomething];
```

U ovom primjeru `someObject` je objekt koji prima poruku, a `doSomething` je poruka koja mu se šalje.

Ako se sjetimo deklaracije i implementacije klase `XYZPerson` možemo vidjeti i kako bi koristili metodu iz te klase na isti način:

```
[somePerson sayHello];
```

Na slici 3.2 može se vidjeti tok izvođenja poziva metode, tj. slanja poruke.

Objekti kao argumenti

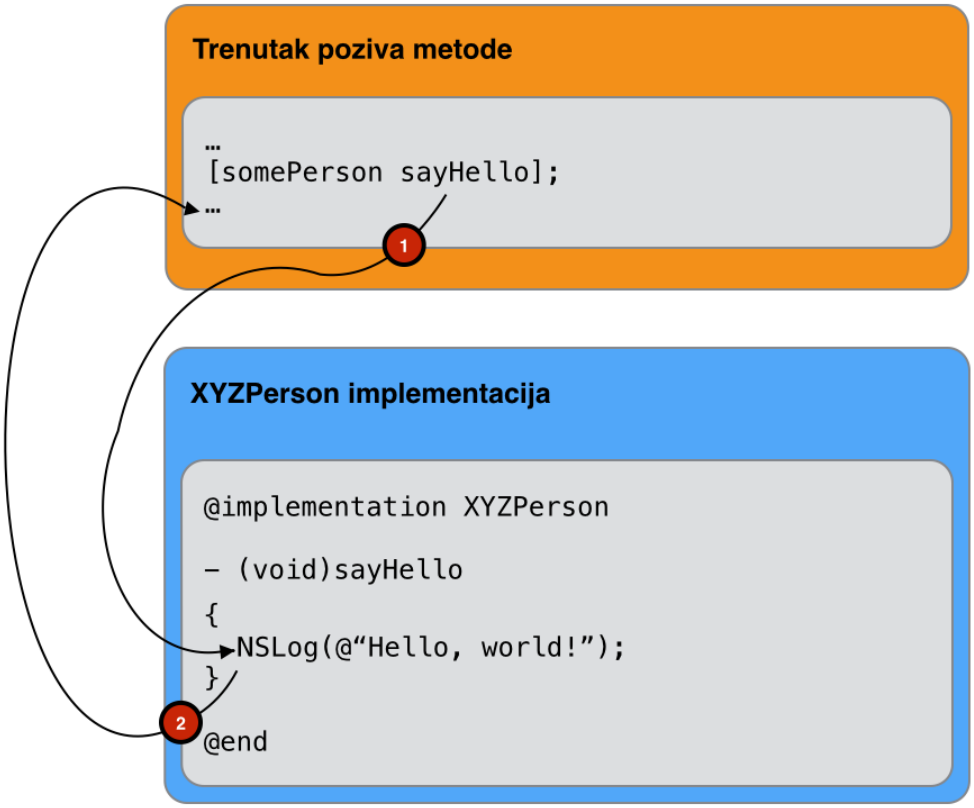
Nekad je korisno slati razne objekte kao argumente pri pozivu metoda. Ako se to treba napraviti jednostavno se koristi pokazivač na određeni objekt. Slijedi primjer gdje se koristi metodu koja ima jedan string objekt kao argument:

```
- (void)saySomething:(NSString *)greeting;
```

Implementacija te metode:

```
- (void)saySomething:(NSString *)greeting
{
  NSLog(@"%@", greeting);
}
```

U tom primjeru varijabla/pokazivač `greeting` u implementaciji metode ponaša se kao lokalna varijabla, ali naravno prisutna je samo u toj metodi. Kad metoda završi s izvršavanjem objekt na koji taj pokazivač pokazuje postojat će i dalje, zato se i koriste pokazivači. Pravilo je da se kreator objekta brine o životnom ciklusu objekta.



Slika 3.2: tok izvođenja poziva metode

U ovom primjeru opet se koristi funkcija `NSLog` i kao što sam već rekao ona je pandan C funkciji `printf()` i u njoj se mogu koristiti razni formati kako bi se ubacili stringovi, brojevi itd. U ovom konkretnom primjeru koristi se @ *token* koji koristi za oznaku supstitucije s objektom. Kad se koristi ta oznaku u `NSLog` funkciji u izvođenju će se pozvati metoda `descriptionWithLocale:` (ako postoji) ili metoda `description` za konkretni objekt koji se koristi. `NSObject` klasa implementira metodu `description` na način da vraća ime klase i memorijsku adresu objekta, ali velik broj Cocoa i Cocoa Touch klasa izmjenjuje tu implementaciju (override) i daju nešto korisnije informacije. Tako klasa `NSString` jednostavno vraća niz znakova koje reprezentira.

Kao i u C-u tako i ovdje metode mogu i ne moraju vraćati vrijednosti. Ako je povratni tip `void` znači da metoda ne vraća ništa, inače vraća objekt određenog tipa. Ako se želi da metoda vraća tip `int` to se radi vrlo jednostavno. U implementaciji metode se naprosto

koristi naredba `return` kao i u C-u.

```
// deklaracija:
- (int)magicNumber;

// implementacija:
- (int)magicNumber
{
  return 42;
}
```

Kod poziva metode povratna vrijednost se može i ne mora spremati u neku varijablu:

```
[someObject magicNumber]; // vrijednost se ne sprema nigdje iako
      ju metoda vraca
int interestingNumber = [someObject magicNumber]; // vrijednost
      se sprema u varijablu
```

Također uz primitivne tipove kao što su `int, float` itd., mogu se i vraćati objekti (točnije pokazivači). Tako primjerice klasa `NSString` ima metodu `uppercaseString` koja vraća verziju sa svim velikim slovima za određeni objekt. Tu metodu bi se moglo ovako koristiti:

```
NSString *helloString = @"Hello, world!";
NSLog(@"%@", [helloString uppercaseString]); // ispisuje se '
      HELLO, WORLD!'
```

Ovdje se vidio primjer poziva metode u samoj `NSLog` funkciji kako bi se odmah ispisala povratna vrijednost. Tako se može ugnježđavati i pozivati metode koliko god želimo.

Kod kreiranja objekata uvijek postoji problem s upravljanjem memorijom (*memory management*) i trebalo bi paziti kakav je životni tok objekta. Npr. ako postoji metodu u kojoj se vrać stvoreni string:

```
- (NSString *)magicString
{
  NSString *stringToReturn = // napravili smo neki string...

  return stringToReturn;
}
```

Ovdje objekt koji je stvoren i za koji se koristi pokazivač `stringToReturn` živi i nakon izvršavanja ove metode iako se taj pokazivač ne može koristiti izvan ove metode. Objective-C kompajler pomoću *Automatic Reference Counting* (automatsko brojanje referenci - ARC) pomaže pri tome i sam se brine da objekt živi sve dok ga originalni kreator

treba. Prije ARC-a to se radilo ručno pomoću `retain` i `release` metoda, ali ja ću u ovom radu pretpostavljati da uvijek koristimo ARC. Naravno, može se i dalje ručno paziti na upravljanje memorijom ako treba, ali u većini slučajeva ARC je odličan.

Slanje poruka sami sebi

Objekti sami sebi mogu slati poruke, tj. pozivati sami svoje metode. Kad se piše implementacija neke metode na raspolaganju je skrivena vrijednost `self`, koja je u biti pokazivač na objekt koji prima trenutnu poruku, tj. nad kojim se poziva određena metoda. Implementacija metode `saySomething:` od prije se može izmijeniti uz korištenje `self`:

```
@implementation XYZPerson
- (void)sayHello
{
  [self saySomething:@"Hello, world!"];
}
- (void)saySomething:(NSString *)greeting
{
  NSLog(@"%@", greeting);
}
@end
```

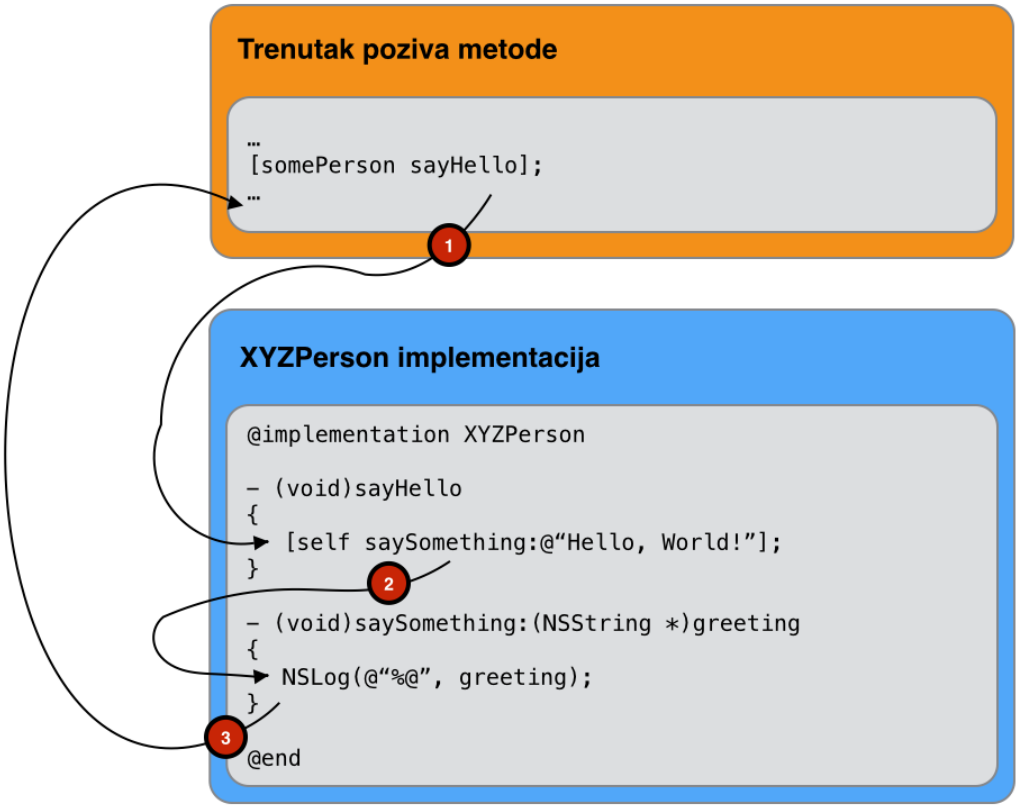
Na taj način se može primjerice naknadno dodati metodu `sayGoodMorning` u kojoj bi se samo pozvalo `saySomething:`. Na slici 3.3 može se vidjeti tok izvođenja s takvom implementacijom.

Postoji još jedan zanimljiv način poziva metoda, a to je koristeći ključnu riječ `super`. Kad se šalje poruka prema `super` to znači da se poruka šalje jednu razinu prema gore prema nadklasi. To se najčešće koristi kad se mijenja implementacija (*override*) neke metode. Na slici 3.4 može se vidjeti tok izvođenja sa `super`.

Dinamično kreiranje objekata

Kad se kreiraju objekti to se uobičajeno radi kombinacijom `alloc` metode i neke od `init` metoda koje ima svaki `NSObject` objekt. Metoda `alloc` je klasna metoda i ona se brine za alokaciju dovoljno memorije za objekt određene klase, čisti taj dio memorije i postavlja sve na nule i vraća pokazivač na sam objekt, koristeći specijalni tip `id` koji znači da vraća neki objekt. Metoda `init` se brine da se naprave sve potrebne inicijalizacije u objektu.

```
+ (id)alloc;
- (id)init;
```



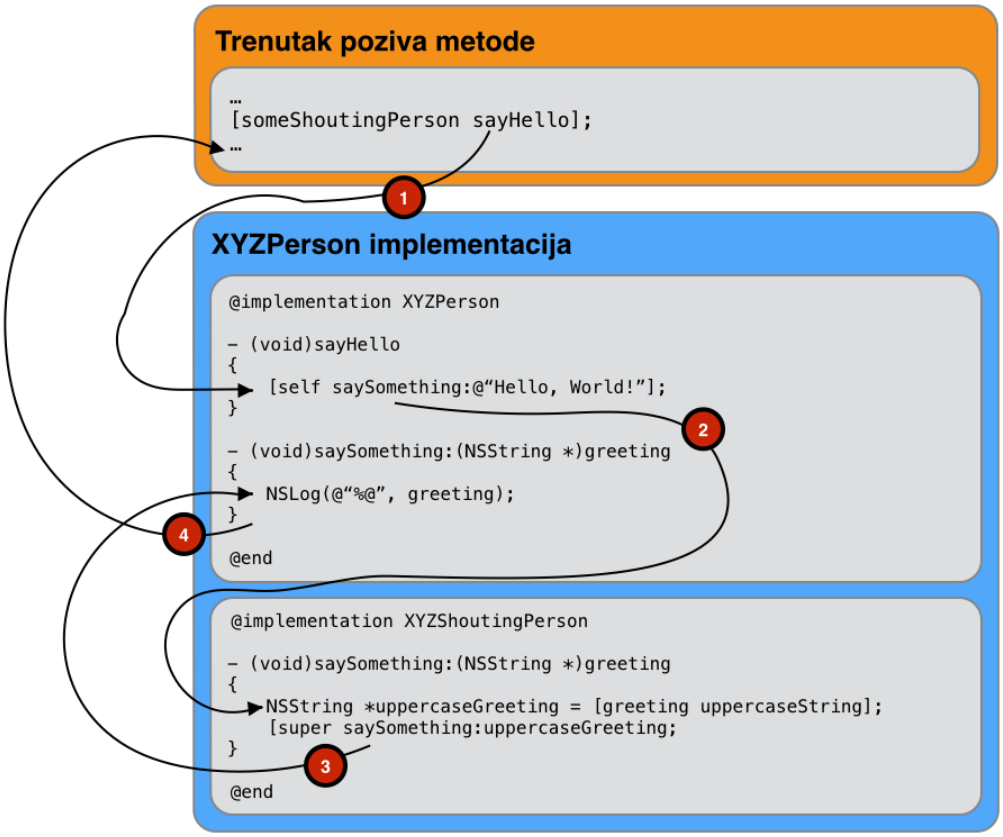
Slika 3.3: Tok izvođenja pri slanju poruke samom sebi

Pošto obje metode vraćaju pokazivače na objekt može ih se ugnjezditi pri kreiranju objekta:

```
NSObject *newObject = [[NSObject alloc] init];
```

Postoje razne verzije `init` metode za specifične inicijalizacije za neke objekte, a možemo i sami implementirati razne metode tog tipa. Nekoliko takvih metoda klase `NSNumber`:

```
- (id)initWithBool:(BOOL)value;
- (id)initWithFloat:(float)value;
- (id)initWithInt:(int)value;
- (id)initWithLong:(long)value;
// koristenje metode:
NSNumber *magicNumber = [[NSNumber alloc] initWithInt:42];
```



Slika 3.4: Tok izvođenja pri slanju poruke sa `super`

Postoji i alternativa kombinaciji metoda za alokaciju i inicijalizaciju, a to su *class factory* metode koje rade i alokaciju i inicijalizaciju odjednom pa se ne trebaju koristiti dvije nego jedna metoda. Slijede primjeri takvih metoda za klasu `NSNumber`:

```
+ (NSNumber *)numberWithBool:(BOOL)value;
+ (NSNumber *)numberWithFloat:(float)value;
+ (NSNumber *)numberWithInt:(int)value;
+ (NSNumber *)numberWithLong:(long)value;
// koristenje metode:
NSNumber *magicNumber = [NSNumber numberWithInt:42];
```

Također se može koristiti i metoda `new` ukoliko se nema potrebe za nekim argumentima pri inicijalizaciji:

```
XYZObject *object = [XYZObject new];
```

```
// efektivno isto kao:
XYZObject *object = [[XYZObject alloc] init];
```

Neke klase omogućuju i korištenje sintakse literala za kreiranje objekata pa se tako primjerice objekti klase `NSString` i `NSNumber` mogu kreirati ovako:

```
NSString *someString = @"Hello, World!";

NSNumber *myBOOL = @YES;
NSNumber *myFloat = @3.14f;
NSNumber *myInt = @42;
NSNumber *myLong = @42L;
NSNumber *myInt = @(84 / 2);
```

Jednakost objekata

Nekad trebamo raditi usporedbe među objektima, a najčešći način usporedbe u programiranju je operator ==. Ako se radi s varijablama tipa `int, double` itd., nema nikakvih problema. Ali kad se upravlja objektima, točnije pokazivačima, malo je drugačija priča. Ovaj primjer koda uspoređuje dva objekta, ali samo govori da li pokazuju na isti objekt:

```
if (firstPerson == secondPerson)
{
    // firstPerson je isti objekt kao secondPerson
}
```

Ako se želi provjeriti da li objekti reprezentiraju iste podatke, koriste se metode kao što je `isEqual:`:

```
if ([firstPerson isEqual:secondPerson])
{
    // firstPerson je identican sa secondPerson
}
```

Objekte se može i uspoređivati na način da se odredi da li jedan reprezentira veću ili manju vrijednost od drugog (za to se u C-u koriste operatori `< i >`). Tako se za osnovne tipove iz `Foundation` frameworka kao što su `NSNumber, NSString i NSDate` koriste `compare:` metode:

```
if ([someDate compare:anotherDate] == NSOrderedAscending)
{
    // someDate je raniji datum nego anotherDate
}
```

Rad s `nil`

Kad se koriste obične skalarne varijable dobro je prilikom same deklaracije i inicijalizirati ih s nekom vrijednošću jer će inače imati neku vrijednost koja je prije bila na toj adresi u memoriji. Kod deklaracije pokazivača na objekte u Objective-C ne treba se raditi nužno inicijalizacija jer će kompajler vrijednost automatski postaviti na `nil`. Ako ne postoji trenutno neka vrijednost s kojom se želi inicijalizirati objekt `nil` je u potpunosti siguran jer je dozvoljeno slanje poruka `nil` objektu. Ako se takvom objektu šalje poruka jednostavno se ništa neće dogoditi. Ako se želi provjeriti da objekt nije `nil` koristi se standardni C operator `!=`:

```
if (somePerson != nil)
{
    // somePerson pokazuje na neki objekt
}
ili jednostavnije
if (somePerson)
{
    // somePerson pokazuje na neki objekt
}
```

Ako je varijabla `somePerson` jednaka `nil` tada je njena logička vrijednost jednaka 0 (*false*), inače je 1 (*true*). Ako se želi provjeriti je li varijabla jednaka `nil` koristi se operator jednakosti ili operator negacije:

```
if (somePerson == nil)
{
    // somePerson ne pokazuje na niti jedan objekt
}
//
if (!somePerson)
{
    // somePerson ne pokazuje na niti jedan objekt
}
```

3.5 Enkapsulacija

Učahurivanje ili enkapsulacija je način koji omogućuje siguran rad s objektima i njihovim unutarnjim podacima koji reprezentiraju objekt. Koriste se metode za čitanje i pisanje kako bi se mijenjala unutarnja reprezentacija objekta.

Get i Set metode

Tzv. *getter* i *setter* metode se koriste kako bi se pristupalo podacima objekta ili ih se mijenjalo. U prije definiranoj klasi `XYZPerson` bila su dva javna svojstva: `firstName` i `lastName`. Ta svojstva omogućuju se da ovako pristupa podacima i mijenjaju podaci:

```
//pristupanje vrijednosti
NSString *firstName = [somePerson firstName];
//postavljanje vrijednosti
[somePerson setFirstName:@"Johnny"];
```

Te metode kompajler automatski stvara, a ako se ne želi primjerice da postoji *setter* metoda onda se u deklaraciji svojstva koristi atribut `readonly`. *Setter* metoda se automatski definira tako da se ispred imena svojstva doda prefiks `set` i nakon toga ide ime svojstva počevši s velikim slovom. *Getter* metoda je naprosto jednaka imenu svojstva, ali se može i promijeniti ime te metode ukoliko tako više odgovara. Često se to radi kad su `BOOL` svojstva u pitanju:

```
@property (readonly, getter=isFinished) BOOL finished;
```

Na taj način se *getter* metodu preimenovala u `isFinished`, a to će omogućiti bolju čitljivost i razumijevanje samog koda.

Može se koristiti i standardna *dot* (točka) notaciju tako da ljudi koji su navikli raditi u C++ mogu i dalje na isti način koristiti getter i setter metode. To je samo *wrapper* (omotač) koji i dalje poziva adekvatne metode.

```
NSString *firstName = somePerson.firstName; // isto kao NSString
      *firstName = [somePerson firstName];
somePerson.firstName = @"Johnny"; // isto kao [somePerson
      setFirstName:@"Johnny"];
```

3.6 Prilagođavanje postojećih klasa

Ponekad umjesto da se radi nova klasa kako bi se proširilo funkcionalnosti neke već postojeće klase koriste se kategorije (*categories*) i ekstenzije (*class extensions*).

Kategorije

Pomoću kategorija vrlo lako se može nekoj već postojećoj klasi dodati neke metode koje bi bile potrebne. Sintaksa definiranja kategorije je slična definiranju same klase, koristi se ključna riječ `@interface`, ali se ne piše koja se klasa nasljeđuje nego se u zagradi piše ime kategorije:

```
@interface ClassName (CategoryName)

@end
```

Svaka metoda koja se doda kroz kategorije bit će dostupna bilo kojem objektu te klase i svim objektima podklasa. Tako da se te metode u biti ne razlikuju od bilo koje druge metode koju je definirala originalna klasa. U primjeru klase `XYZPerson` moglo bi se kroz kategoriju dodati metodu koja za objekt klase `XYZPerson` omogućuje da se dobije spoj imena i prezimena osobe. Kategorije se definiraju u odvojenim datotekama pa bi tako header datoteka bila *XYZPerson+XYZPersonNameDisplayAdditions.h* koja sadrži deklaracije:

```
#import "XYZPerson.h"

@interface XYZPerson (XYZPersonNameDisplayAdditions)

- (NSString *)lastNameFirstNameString;

@end
```

Implementacija te metode:

```
// XYZPerson+XYZPersonNameDisplayAdditions.m datoteka:
#import "XYZPerson+XYZPersonNameDisplayAdditions.h"

@implementation XYZPerson (XYZPersonNameDisplayAdditions)

- (NSString *)lastNameFirstNameString
{
  return [NSString stringWithFormat:@"%@, %@", self.lastName,
        self.firstName];
}
@end
```

Tako definirana metoda preko kategorije se može vrlo lako koristiti:

```
#import "XYZPerson+XYZPersonNameDisplayAdditions.h"

@implementation SomeObject
- (void)someMethod
{
  XYZPerson *person = [[XYZPerson alloc] initWithFirstName:@"John
        " lastName:@"Doe"];
```

```
    XYZShoutingPerson *shoutingPerson = [[XYZShoutingPerson alloc]
            initWithFirstName:@"Monica" lastName:@"Robinson"];
    NSLog(@"The two people are %@ and %@", [person
            lastNameFirstNameString], [shoutingPerson
            lastNameFirstNameString]);
}
@end
```

Kao što se vidi kategorije su jako praktične jer se ne treba raditi nova klasa koja nasljeđuje neku već postojeću klasu, a vrlo lako se može proširiti funkcionalnosti već postojeće klase. Treba paziti kod imenovanja takvih metoda, jer ako postoji u kategoriji metoda isto deklarirana kao i u originalnoj klasi nije definirano koja će se od metoda koristiti. Da bi se izbjegao taj problem dobro je imenu metode dodati prefiks. Tako primjerice kod metode `sortDescriptorWithKey:ascending:` klase `NSSortDescriptor`, metoda u kategoriji (ako se želi isto takvo opisno ime) bi izgledala ovako:

```
@interface NSSortDescriptor (XYZAdditions)
+ (id)xyz_sortDescriptorWithKey:(NSString *)key ascending:(BOOL)
      ascending;
@end
// koristenje:
NSSortDescriptor *descriptor = [NSSortDescriptor
      xyz_sortDescriptorWithKey:@"name" ascending:YES];
```

Ekstenzije klase

Ekstenzije klase su slične kategorijama, ali može ih dodavati jedino ako se ima pristup izvornom kodu klase pri kompajliranju, dok kod kategorija to nije bitno. Dakle, ne mogu se raditi ekstenzije za klase iz Cocoa ili Cocoa Touch jer nemamo njihov izvorni kod.

Sintaksa je jednostavna, slično kao i kod definicije kategorije, samo se ništa ne piše u zagradu. Pomoću ekstenzija mogu se dodavati i metode i svojstva.

```
@interface XYZPerson ()
@property NSObject *extraProperty;
@end
```

Ekstenzije su korisne za skrivanje nekih privatnih informacija i funkcionalnosti koje bi koristili u dijelu implementacije klase. Tako da se ekstenzije tamo i pišu.

```
@interface XYZPerson ()
@property (readwrite) NSString *uniqueIdentifier;
@end
```

```
@implementation XYZPerson
...
@end
```

3.7 Vrijednosti i kolekcije

Iako je Objective-C objektno orijentiran programski jezik on je i nadskup C-a, tako da se mogu koristiti i standardni skalarni tipovi `int, float, char` u Objective-C kodu. U Cocoa i Cocoa Touch postoje također i `NSInteger, NSUInteger, CGFloat`, koji imaju drukčije definicije ovisno o arhitekturi sustava za koji se rade aplikacije.

Moguće je koristiti standardna polja iz C-a unutar Objective-C koda, ali praktičnije i jednostavnije je koristiti klase kao `NSArray` i `NSDictionary` u aplikacijama. I te klase mogu sadržavati ostale Objective-C objekte.

Osnovni tipovi

Osnovni tipovi i operatori iz C-a:

```
int someInteger = 42;
float someFloatingPointNumber = 3.1415;
double someDoublePrecisionFloatingPointNumber = 6.02214199e23;
// operatori:
int someInteger = 42;
someInteger++; // someInteger == 43
int anotherInteger = 64;
anotherInteger--; // anotherInteger == 63
anotherInteger *= 2;  // anotherInteger == 126
```

Tipovi iz C-a se mogu normalno koristiti kao svojstva u klasama:

```
@interface XYZCalculator : NSObject
@property double currentValue;
@end
// //
@implementation XYZCalculator
- (void)increment
{
  self.currentValue++;
}
- (void)decrement
```

```
{
  self.currentValue--;
}
- (void)multiplyBy:(double)factor
{
  self.currentValue *= factor;
}
```

Postoje i osnovni Objective-C tipovi: `BOOL`(`YES` i `NO`), `NSInteger, CGFloat` itd. Tipovi kao `NSInteger, NSUInteger` su definirani ovisno o arhitekturi za koju se radi aplikacija. Ako je okruženje 32-bitno onda predstavljaju 32-bitne cijele ili pozitivne prirodne brojeve, a ako je okruženje 64-bitno onda su 64-bitni.

Stringovi su jako česta pojava pa tako se i u Objective-C često koristi `NSString` koji se može kreirati i mijenjati na razne načine:

```
NSString *firstString = [[NSString alloc] initWithCString:"Hello
      World!" encoding:NSUTF8StringEncoding];
NSString *secondString = [NSString stringWithCString:"Hello World
      !" encoding:NSUTF8StringEncoding];
NSString *thirdString = @"Hello World!";
//
NSString *name = @"John";
name = [name stringByAppendingString:@"ny"]; // vraca novi string
       objekt
//
NSMutableString *name = [NSMutableString stringWithString:@"John"
      ];
[name appendString:@"ny"];    // isti objekt, ali sada sadrzi "
      Johnny"
//
int magicNumber = ...
NSString *magicString = [NSString stringWithFormat:@"The magic
      number is %i",
   magicNumber];
```

Klasa `NSNumber` se koristi za reprezentaciju osnovnih skalarnih tipova iz C-a (mogu se koristiti i tipovi `NSInteger, CGFLoat` itd.:

```
NSNumber *magicNumber = [[NSNumber alloc] initWithInt:42];
NSNumber *unsignedNumber = [[NSNumber alloc] initWithUnsignedInt
      :42u];
NSNumber *longNumber = [[NSNumber alloc] initWithLong:42l];
NSNumber *boolNumber = [[NSNumber alloc] initWithBOOL:YES];
```

```
NSNumber *simpleFloat = [NSNumber numberWithFloat:3.14f];
NSNumber *betterDouble = [NSNumber numberWithDouble
      :3.1415926535];
NSNumber *someChar = [NSNumber numberWithChar:'T'];
//kreiranje pomocu literal sintakse:
NSNumber *magicNumber = @42;
NSNumber *unsignedNumber = @42u;
NSNumber *longNumber = @42l;
NSNumber *boolNumber = @YES;
NSNumber *simpleFloat = @3.14f;
NSNumber *betterDouble = @3.1415926535;
NSNumber *someChar = @'T';
```

Kad se koristi neki objekt klase `NSNumber` lako se mogu koristiti metode pristupa (*accessor methods*) kako bi se pristupilo određenoj vrijednosti:

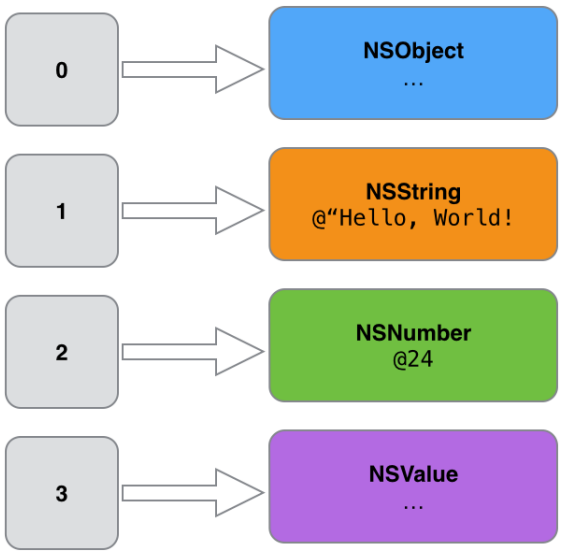
```
int scalarMagic = [magicNumber intValue];
unsigned int scalarUnsigned = [unsignedNumber unsignedIntValue];
long scalarLong = [longNumber longValue];
BOOL scalarBool = [boolNumber boolValue];
float scalarSimpleFloat = [simpleFloat floatValue];
double scalarBetterDouble = [betterDouble doubleValue];
char scalarChar = [someChar charValue];
```

Kolekcije

Kao što sam već rekao, moguće je koristiti standardna polja iz C-a unutar Objective-C koda, ali praktičnije i jednostavnije je u aplikacijama koristiti klase iz Cocoa i Cocoa Touch kao `NSArray`, `NSSet` i `NSDictionary`. Te klase mogu sadržavati ostale Objective-C objekte. Ako se želi u neku od kolekcija dodati skalarnu vrijednost potrebno je koristiti `NSNumber` ili `NSValue` tip i u njega spremiti vrijednost. Osnovni tipovi `NSArray`, `NSSet` i `NSDictionary` su *immutable*, tj. ne možemo mijenjati njihov sadržaj nakon kreiranja samog objekta kolekcije, ali postoje i *mutable*, tj. promjenjive verzije tih klasa u kojima se mogu dodavati ili uklanjati objekti.

NSArray

Klasu `NSArray` se koristi za reprezentaciju poredane kolekcije objekata. Jedini uvjet je da je svaki element kolekcije Objective-C objekt, nije bitno koje su klase ti objekti, tj. može u jednoj kolekciji biti više primjera različitih klasa. Elementi su poredani jedan za drugim i počinju s indeksom 0 (slika 3.5).



Slika 3.5: polje Objective-C objekata

Postoji nekoliko metoda pomoću kojih se može kreirati objekt tipa `NSArray`, od kojih se neke koriste tako da se nabrajaju elementi za polje i završava s *nil* kao oznakom kraja polja.

```
+ (id)arrayWithObject:(id)anObject;
+ (id)arrayWithObjects:(id)firstObject, ...;
- (id)initWithObjects:(id)firstObject, ...;
//
NSArray *someArray = [NSArray arrayWithObjects:someObject,
      someString, someNumber, someValue, nil];
// literal sintaksa:
NSArray *someArray = @[firstObject, secondObject, thirdObject];
```

Postoji mnogo načina kako se može pregledavati polje ili tražiti elemente u polju. Neki od tih načina slijede:

```
// dobiti broj elemenata u polju
NSUInteger numberOfItems = [someArray count];
// vidjeti da li postoji neki element u polju
```

```
if ([someArray containsObject:someString])
{
  ...
}
// pristup elementu na odredenom indeksu
if ([someArray count] > 0)
{
  NSLog(@"First item is: %@", [someArray objectAtIndex:0]);
}
```

Vrlo jednostavno se mogu i sortirati polja koristeći razne metode kao usporednu funkciju u sortiranju. Slijedi primjer gdje se koristi standardnu `compare:` metodu koja radi leksikografsku usporedbu:

```
NSArray *unsortedStrings = @[@"gammaString", @"alphaString", @"
    betaString"];
NSArray *sortedStrings = [unsortedStrings
    sortedArrayUsingSelector:@selector(compare:)];
```

Može se koristiti i `NSMutableArray` klasa i na taj način je omogućeno dodavanje element po element u polje, mijenjanje ili micanje iz polja ako želimo:

```
NSMutableArray *mutableArray = [NSMutableArray array];
[mutableArray addObject:@"gamma"];
[mutableArray addObject:@"alpha"];
[mutableArray addObject:@"beta"];
[mutableArray replaceObjectAtIndex:0 withObject:@"epsilon"];
```

Takav tip polja se može sortirati bez da se kreira novo sortirano polje:

```
[mutableArray sortUsingSelector:@selector(caseInsensitiveCompare
    :)];
```

NSSet

`NSSet` je kolekcija slična polju, ali ovdje poredak nije bitan, bitno je samo da su objekti u kolekciji jedinstveni. S obzirom da poredak nije bitan ovaj tip kolekcije je puno efikasniji ako se želi provjeravati da li se neki objekt nalazi u kolekciji.

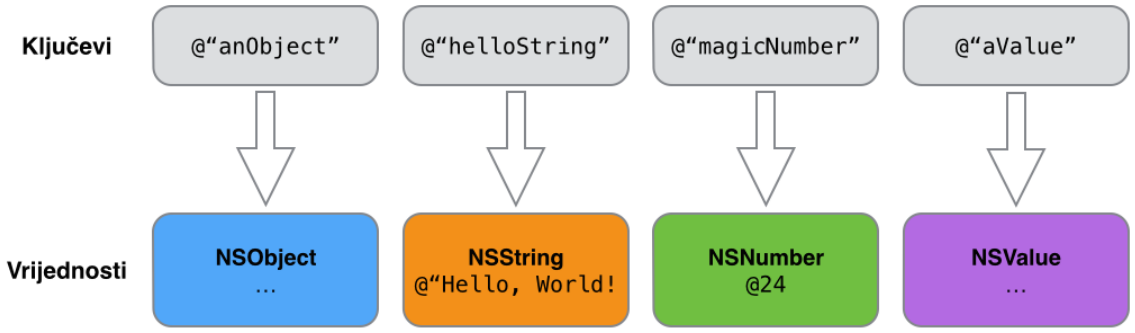
Nekoliko načina kreiranja objekta tog tipa:

```
NSSet *simpleSet = [NSSet setWithObjects:@"Hello, World!", @42,
    aValue, anObject, nil];
//
NSNumber *number = @42;
```

```
NSSet *numberSet = [NSSet setWithObjects:number, number, number,
      number, nil];
// numberSet sadrzi samo jedan element - zbog pravila
      jedinstvenosti
```

NSDictionary

`NSDictionary` (rječnik) je kolekcija koja sadrži objekte na način da ima relaciju (ključ -> objekt) - slika 3.6. Dakle ovdje je slična situacija kao kod polja, samo je ovdje indeks ključ koji može biti bilo koji objekt, ali uobičajena praksa je korištenje stringova za ključeve.



Slika 3.6: relacija ključ -> objekt

Kreiranje objekata i pretraživanje:

```
NSDictionary *dictionary = [NSDictionary
      dictionaryWithObjectsAndKeys: someObject, @"anObject", @"
      Hello, World!", @"helloString" @42, @"magicNumber",
      someValue, @"aValue", nil];
// literal sintaksa:
NSDictionary *dictionary = @{ @"anObject" : someObject, @"
      helloString" : @"Hello, World!", @"magicNumber" : @42, @"
      aValue" : someValue };
//
NSNumber *storedNumber = [dictionary objectForKey:@"magicNumber"
      ];
NSNumber *storedNumber = dictionary[@"magicNumber"];
```

Postoji i *mutable* verzija - `NSMutableDictionary`.

```
[dictionary setObject:@"another string" forKey:@"secondString"];
[dictionary removeObjectForKey:@"anObject"];
```

Prolazak kroz kolekcije

Vrlo jednostavno se može prolaziti kroz kolekcije koje je se koriste kako bi se nešto radilo s objektima spremljenim u kolekcijama.

```
for (<Type> <variable> in <collection>)
{
   ...
}

// polje
for (id eachObject in array)
{
  NSLog(@"Object: %@", eachObject);
}

// dictionary
for (NSString *eachKey in dictionary)
{
  id object = dictionary[eachKey];
  NSLog(@"Object: %@ for key: %@", object, eachKey);
}

// enumerator
for (id eachObject in [array reverseObjectEnumerator])
{
   ...
}
```

3.8 Blokovi

Kad se rade aplikacije koriste se klase i objekti tih klasa kako bi se izgradila aplikacije. Ponekad je korisno imati neki oblik “radilice”, koja će odraditi neki posao i to je sve. Tu dolaze blokovi (*block*) koji omogućuju da se komadi/blokovi koda koriste u metodama kao argumenti gdje se ponašaju kao i ostali argumenti u metodama.

Sintaksa blokova

Sintaksa je vrlo jednostavna pa se tako blok može kreirati koristeći blok literal koji počinje znakom ^. Sljedeći blok ne vraća nikakvu vrijednost i ne prima argumente:

```
^{
     NSLog(@"This is a block");
}
```

Može se i deklarirati varijablu koja bi držala taj blok, slično kao što su C-u postoje pokazivači na funkcije:

```
void (^simpleBlock)(void); // nema argumenata i ne vraca nista
//
simpleBlock = ^{
  NSLog(@"This is a block");
};
// poziv:
simpleBlock();
```

Argumenti i povratne vrijednosti

Blokovi mogu i primati argumente i vraćati vrijednosti kao i obične metode. Sljedeći blok prima dva broja i vraća njihov umnožak:

```
double (^multiplyTwoValues)(double, double); // deklaracija
// blok literal - definicija konkretnog bloka
^ (double firstValue, double secondValue)
{
  return firstValue * secondValue;
}
// ili s eksplicitnim povratnim tipom
^ double (double firstValue, double secondValue) {
    return firstValue * secondValue;
}
```

Kad se deklarira i definira blok, može ga se koristiti kao običnu funkciju:

```
double (^multiplyTwoValues)(double, double) = ^(double firstValue
      , double secondValue) {
  return firstValue * secondValue;
};
double result = multiplyTwoValues(2,4);
NSLog(@"The result is %f", result);
```

Vrijednosti van okruženja bloka

Blok također može koristiti vrijednosti koje se nalaze u okruženju (*scope*) u kojem je blok. Ako se blok nalazi u nekoj metodi mogu se koristiti varijable van bloka:

```
- (void)testMethod
{
  int anInteger = 42;
  void (^testBlock)(void) = ^{
  NSLog(@"Integer is: %i", anInteger);
  };
  testBlock();
}
```

Vrijednost varijable `anInteger` se uzima pri definiciji bloka, tako da ako se nakon bloka promijeni vrijednost u toj varijabli neće biti utjecaja na vrijednost u bloku:

```
int anInteger = 42;
void (^testBlock)(void) = ^{
  NSLog(@"Integer is: %i", anInteger);
};
anInteger = 84;
testBlock(); // ispisuje: Integer is: 42
```

Ako se želi omogućiti promjene vrijednosti onda se treba koristiti `__block` varijable tako da varijabla, a ne samo vrijednost, živi i u metodi i u bloku. Primjer od maloprije:

```
__block int anInteger = 42;
void (^testBlock)(void) = ^{
  NSLog(@"Integer is: %i", anInteger);
};
anInteger = 84;
testBlock(); // ispisuje: Integer is: 84
```

A može se i unutar bloka mijenjati originalnu varijablu:

```
__block int anInteger = 42;
void (^testBlock)(void) = ^{
  NSLog(@"Integer is: %i", anInteger);
  anInteger = 100;
};
testBlock(); // ispisuje: Integer is: 42
NSLog(@"Value of original variable is now: %i", anInteger); //
      ispisuje: Value of original variable is now: 100
```

Blokovi kao argumenti

Blokovi se ipak najčešće koriste kao argumenti koji se šalju nekoj metodi ili funkciji. Često se koriste uz *Grand Central Dispatch* kako bi se neki zadatak izvršio u pozadini, često za primjenu bloka koda pri prolasku kroz kolekciju itd. Koriste se također i kao povratni pozivi (*callbacks*) koji se koriste kako bi se dobio povratni poziv pri završetku neke metode ili zadatka. Primjer:

```
- (IBAction)fetchRemoteInformation:(id)sender
{
  [self showProgressIndicator];
  XYZWebTask *task = ...
  [task beginTaskWithCallbackBlock:^{
    [self hideProgressIndicator]; // blok se izvrsi tek kad se
          task zavrsi
  }];
}
// deklaracija gornje metode:
- (void)beginTaskWithCallbackBlock:(void (^)(void))callbackBlock;
// definicija:
- (void)beginTaskWithCallbackBlock:(void (^)(void))callbackBlock
{
  ...
  callbackBlock();
}
```

Enumeracija

Blokovi su jako korisni u kolekcijama kad se želi proći kroz cijelu kolekciju i napraviti neki zadatak za svaki objekt unutar kolekcije. Primjerice za `NSArray`:

```
- (void)enumerateObjectsUsingBlock:(void (^)(id obj, NSUInteger
      idx, BOOL *stop))block;
NSArray *array = ...
[array enumerateObjectsUsingBlock:^ (id obj, NSUInteger idx, BOOL
       *stop) {
  NSLog(@"Object at index %lu is %@", idx, obj);
}];
// mozemo pomocu treceg argumenta zaustavit izvodenje bloka:
[array enumerateObjectsUsingBlock:^ (id obj, NSUInteger idx, BOOL
       *stop) {
  if (...)
```

```
  {
    *stop = YES;
  }
}];
```

Blokovi u Grand Central Dispatch

Blokovi se često koriste uz Grand Central Dispatch (GCD) kako bi se blokovi koda slali na izvođenje u razne redove (queues), na sinkrono ili asinkrono izvođenje.

```
dispatch_queue_t queue = dispatch_get_global_queue(
      DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
//
dispatch_async(queue, ^{
  NSLog(@"Block for asynchronous execution");
});
```

3.9 Greške

Moramo pretpostaviti da će se ponekad javiti koja greška u radu aplikacije. Može se dogoditi da je nestalo mjesta na disku, da nema pristupa internetu (a radi se neki networking). Zato bi se trebalo reagirati na takve greške, planirati ih i probati što je bolje riješiti takve problematične situacije. Mogu se dogoditi i iznimke (*exceptions*), ali pretpostavljamo da se njih rješavamo prilikom samog razvoja aplikacije.

NSError

`NSError` je klasa koja se koristi za reprezentaciju grešaka prilikom rada aplikacije. Primjerice kad se pozove ova metoda (*delegate callback*) zna se da je došlo do greške prilikom konekcije i imamo priliku reagirati:

```
- (void)connection:(NSURLConnection *)connection didFailWithError
      :(NSError *)error;
```

Možemo i sami kreirati `NSError` objekt i koristiti ga kao argument u pozivanju metoda kako bismo dobili povratnu informaciju da li se dogodila neka greška ili ne. Ako se zapisuju neki podaci spremljeni u `NSData` koristi se ova metoda i šalje se objekt kao referenca (**):

```
- (BOOL)writeToURL:(NSURL *)aURL options:(NSDataWritingOptions)
      mask error:(NSError **)errorPtr;
```

```
// koristenje:
NSError *anyError;
BOOL success = [receivedData writeToURL:someLocalFileURL options
      :0 error:&anyError];
if (!success)
{
  NSLog(@"Write failed with error: %@", anyError);
  // pokazati gresku korisniku
}
```

Ako se dogodi neka greška trebalo bi reagirati i kroz što bolje korisničko iskustvo riješiti samu grešku. Ako je moguće ponovno pokušati napraviti zadatak bez greške. A može se i pomoću klase `UIAlertView` korisniku pomoću popup prozora prenijeti informaciju o greški.

3.10 Konvencije

Kad se radi s klasama iz raznih frameworka iz SDK lako je za primijetiti da je sve vrlo jednostavno za čitanje, gotovo da nije ni potrebno čitati detalje dokumentacije kako bi se vidjelo što određena metoda radi. To je zato jer se u Objective-C, Cocoa i Cocoa Touch drži do konvencija i to dovodi do velike razine čitljivosti i razumljivosti samog koda.

Imena klasa

Kod imena klasa koristi se *camel case* oblik, npr. `MyNewClass`. Imena klasa moraju biti jedinstvena kroz cijelu aplikaciju. Kako bismo bili sigurni da imamo jedinstvena imena praksa je dodati prefikse ispred imena klase, npr. `BWLocation` u mojoj aplikaciji "BackWay". Tako Cocoa i Cocoa Touch klase imaju prefikse `NS` i `UI`. Slijede neki primjeri prefiksa za Cocoa i Cocoa Touch:

prefiks	framework
NS	Foundation (OS X i iOS), Application Kit (OS X)
UI	UIKit (iOS)
AB	Address Book
CA	Core Animation
CI	Core Image

Kad se određuju imena klasa trebalo bi dobro razmisliti i što bolje kroz ime objasniti što klasa reprezentira. Primjeri iz Cocoa i Cocoa Touch: `NSWindow`, `CAAnimation`, `NSWindowController`, `NSManagedObjectContext`, `UINavigationController`.

Imena metoda

Imena metoda unutar neke klase trebaju biti jedinstvena, ali trebaju i jasno dočarati samim imenom što metoda radi. Ime metode počinje malim slovom, a nastavlja se koristeći *camel case*. Primjeri:

- `length`
- `characterAtIndex:`
- `lengthOfBytesUsingEncoding:`
- `substringFromIndex:`
- `writeToURL:atomically:encoding:error:`
- `enumerateSubstringsInRange:options:usingBlock:`

Imena varijabli

Lokalne varijable moraju biti jedinstvene u svom okruženju, kao i u C-u:

```
- (void)someMethod
{
  int interestingNumber = 42;
  ...
  int interestingNumber = 44; // NIJE DOPUSTENO!!
}
```

Iako se može u okruženju unutar nekog okruženja deklarirati varijablu istog imena, trebalo bi to izbjegavati jer otežava čitljivost i razumljivost koda:

```
- (void)someMethod
{
  int interestingNumber = 42;
  ...
  for (NSNumber *eachNumber in array)
  {
    int interestingNumber = [eachNumber intValue]; // KOMPLICIRA
        CITLJIVOST
  }
}
```

Ostale konvencije

Za metode pristupa svojstvima (*accessor methods*), primjerice za svojstvo `firstName`, getter metoda bi trebala biti `firstName`, a setter metoda `setFirstName`. Ukoliko je svojstvo BOOL tipa onda bi getter metoda trebala biti npr. `isPaused`.

Metode inicijalizacije, alokacije, kreiranja objekata trebaju pratiti konvenciju:

```
NSMutableArray *array = [[NSMutableArray alloc] init];
NSMutableArray *array = [NSMutableArray new];
NSMutableArray *array = [NSMutableArray array];
```

Poglavlje 4

Razvojni proces

Razvoj svakog softvera ima neki svoj tijek. Postoje razne metode i tehnike koje se koriste pri razvoju softvera, ali ja se ne držim čvrsto ni jedne metode, više se držim svog osjećaja. Možda je takav slučaj jer radim na svojim projektima, a ne u nekoj kompaniji pa mogu slobodno raditi kako meni odgovara. U ovom poglavlju će biti riječi o tipičnom razvojnom procesu neke iOS aplikacije, od početka razvoja pa sve do objave na App Storeu.

Može se reći da se početak razvoja neke iOS aplikacije sastoji od pripreme razvojnog okruženja (Mac računalo, Xcode itd.), detaljnijeg razvoja ideje same aplikacije i razrade dizajna (struktura, sučelje, interakcija itd.).

4.1 Uvod i osnove

Za početak je potrebno imati pripremljeno razvojno okruženje i ostale materijale koji su potrebni za cijeli razvoj (papiri, ostali softveri nevezani za programiranje itd).

Priprema

Većina Mac računala već imaju instalirane sve aplikacije potrebne za razvoj. Ukoliko to nije slučaj, potrebno je besplatno skinuti Appleov IDE Xcode iz Mac App Storea koji sadrži sve alate, kompajlere i cijeli iOS SDK. Xcode ima i već pripremljenu cijelu dokumentaciju za Mac OS i iOS tako da je i cijela dokumentacija automatski dostupna.

Početak

Ovisno o tipu projekta koji se radi, bila to igra ili općenito neka aplikacija, pri početku razvoja kreira se novi projekt koji će sadržavati sve resurse (kod, grafike itd.) za aplikaciju koja se radi. Odlično je to što se ne treba skakati iz programa u program nego je sve

dostupno unutar Xcodea i može se ugodno razvijati aplikacije. Pri kreiranju novog projekta može se birati neke već definirane kosture (*templates*) koji donose već napravljen dio aplikacije. Može se odabrati primjerice *Empty Application* (najrudimentarniji template), *Single View Application*, *Sprite Kit Game* itd. Naravno ako se odabere primjerice praznu aplikaciju kao kostur, to ne znači da se ne može iz toga napraviti igru i obratno. Ti kosturi samo služe kako bismo već imali neke stvari posložene. Pri kreiranju projekta može se automatski napraviti i git[1] repozitorij ili se povezati na neki git repozitorij na nekom serveru.

4.2 Struktura aplikacije

Kad odlučim krenuti u realizaciju neke ideje u konkretnu aplikaciju prvo krenem s pisanjem opisa aplikacije (funkcionalnosti aplikacije) i jednostavnim skicama na običnom papiru ili App Sketchbook bilježnici - slika 8.2. Uz obično pisanje i skiciranje na papiru koristim i osnovnu "Notes" aplikaciju za pisanje nekih ideja koje mi svakodnevno padnu na pamet kad nisam direktno u razvoju. Ne volim koristiti neke "ozbiljne" softvere za organizaciju i upravljanje projektima, ali koristim web aplikaciju Asana[2] kako bih si posložio projekte na kojima radim, koje funkcionalnosti mislim implementirati i što planiram u budućim verzijama aplikacija.

Smatram da je jako važno u početku razvoja biti fokusiran i koncentirati se samo na najbitnije aspekte i funkcionalnosti aplikacije. Jer ako si zadamo previše nepotrebnog posla samo ćemo zakomplicirati aplikaciju i sam razvoj što u velikom broju slučajeva ne donese ništa dobrog ni korisnog.

Svi ti dijelovi razvoja nisu uvijek isti ni istog poretka. I ovisno o projektu nije uvijek svaki dio jednak što i daje čar samom razvoju aplikacija jer svaki projekt donosi nešto novo i različito.

Jako je važno postaviti si sljedeća pitanja i odgovoriti na njih.

- **Tko će koristiti aplikaciju?** - bitno je znati da li se radi aplikaciju za djecu, trudnice ili muškarce koji vole jesti meso.
- **Koja je svrha vaše aplikacije?** - bitno je jasno definirati koji je cilj i svrha aplikacije koja se radi. Odlično je i koncentrirati se na jednu stvar i tu jednu stvar napraviti perfektno jer će tako korisnici biti motivirani za korištenje aplikacije.
- **Koji problem rješava aplikacija?** - nije dobro raditi aplikacije koje rješavanju nepostojeće probleme. Treba se fokusirati na rješavanje jednog problema i riješiti ga što je moguće bolje.

[1] `http://en.wikipedia.org/wiki/Git_(software)`
[2] `http://asana.com`

- **Kakav sadržaj će imati vaša aplikacija?** - kakav sadržaj će aplikacija koristiti i prikazivati. Jako bitno za dizajn sučelja i korisničkog iskustva.

Dizajn, arhitektura i struktura

Svaka aplikacija ima glavne komponente, neke manje neke veće. Važno je odmah u početku napraviti te podjele kako bi se lakše mogle raditi konkretne implementacije kasnije. Jasnim odvajanjem komponenti u početku pri dizajniranju same strukture aplikacije dobivaju dugoročne koristi, pri implementaciji funkcionalnosti, budućih novih verzija, ispravaka bugova itd.

Ja osobno strukturu aplikacije, komponente i povezanosti najradije crtam i pišem na papir. To mi je najjednostavnija metoda i meni osobno najefikasnija. Postoje naravno razni softveri koji se mogu koristiti za takve stvari kao i ostale dijelove dizajniranja (npr. sučelja). Možda bi mi bilo drukčije ako bih radio na nekim drugim projektima, ali na dosadašnjim mojim projektima me ovakav način odlično služio.

Dizajn korisničkog sučelja

Nakon što imam većinu strukture aplikacije osmišljenu krećem na dizajn korisničkog sučelja. Ovdje sam i dalje na papiru i olovci i skiciram elemente sučelja i slažem razne ekrane i razne slučajeve pri korištenju aplikacije. Papir i olovka mi omogućuju da jednostavno radim promjene i iznesem razne ideje i brzo mogu vidjeti kako bi nešto izgledalo.

Dizajn interakcije

Interakcija kod uređaja osjetljivih na dodir je potpuno različita od one na tradicionalnim osobnim računalima. Ovdje su glavne interakcije razne geste koje korisnik radi pri interakciji s ekranom svog uređaja. Jako je bitno da se ne izmišljavaju neka nelogična ponašanja za određene geste jer su korisnici iOS-a naviknuti da će određenim gestama imati određene rezultate. Ako se držimo već postojećih pravila korisnici će automatski znati kako koristiti aplikaciju, a to i treba biti cilj, lako korištenje aplikacije.

U ovom dijelu na već postojeće skice korisničkog sučelja dodajem informacije kako će korisnik koristiti elemente sučelja, što će se događati ako se tapne neki gumb itd. Tako da kad se krene s implementacijom kratkim pogledom na skice sučelja mogu vidjeti kako i gdje spojiti pozive metoda i reakcije na razne metode.

4.3 Implementacija

Ono po čemu se developeri najviše razlikuju je da li znaju sami osmisliti cijelu aplikaciju (od ideje, dizajna pa implementacije) ili su "koderi" koji samo implementiraju zadane zadatke. To je naravno moje osobno mišljenje. Najlakše je dobiti zahtjev za nekom funkcionalnošću ili detaljnu specifikaciju i samo pisati kod. Rješavanje problema kao što su dizajn jasnog korisničkog sučelja i realizacija odličnog korisničkog iskustva su stvari koje developere izdvajaju od ostale mase developera.

Ukoliko se sve stavke prije same implementacije naprave dobro i ne krene se s implementacijom čim ideja padne na pamet, dio same aplikacije će teći puno lakše i bezbolnije. Loše dizajnirana struktura aplikacije se teško može popraviti implementacijom, a dobra struktura aplikacije se teško može pokvariti implementacijom. Tako da je moj savjet da se ima strpljena pri početnim fazama razvoja aplikacije kako bi imali manje problema pri kasnijim fazama.

Korištenje podataka

Svaka aplikacija na neki način koristi neke tipove podataka i upravlja s njima. Na nama je da odlučimo na koji način ćemo reprezentirati informacije. Različiti tipovi podataka se koriste za spremanje teksta, a različiti za spremanje slika itd. Jako je važno modele dizajnirati tako da adekvatno reprezentiraju ono što smo zamislili i omoguće nam jasan pogled na strukturu podataka u cijeloj aplikaciji. Postoje različite osnovne strukture podataka (polja, redovi itd.) i odabirom pravih struktura može se dobiti puno na performansama ili jednostavnosti koda i implementacije. Postoje i već definirane klase koje odlično rješavaju neke zadatke i probleme pa nema potrebe raditi svoje klase nego se može u potpunosti koristiti postojeće klase i imati koristi od njih. Također bitno je i vidjeti od kuda će podaci dolaziti i kako će se spremati. Neki podaci će se dobivati interakcijom korisnika i sučelja, neki primjerice korištenjem kamere uređaja. Ako će se podaci trajno spremati u aplikaciju možda će se koristiti neka baza podataka, dokumenti ili možda Core Data framework.

Korištenje design patterns

Za neke već postojeće probleme postoje jasna i uobičajena rješenja. Tako da se može dosta sigurno koristiti već postojeća rješenja i *design patterns* (neka uobičajena pravila ili obrasci) kako bi se što efikasnije iskoristio framework koji se koristi. Korištenje takvih patterna omogućuje da se lako mogu raditi izmjene u aplikaciji bez da se poremeti ostatak aplikacije.

MVC (eng. Model-View-Controller) paradigma je glavna za svaku iOS aplikaciju. Objekti u aplikaciji mogu imati jednu od tri uloga: *model* (podaci), *view* (prikaz) ili *con-*

troller (mozak, radilica, upravljaju prikazom). Ta paradigma daje čistu podjelu u strukturi aplikacije.

Delegation (Delegacija) je dosta koristan pattern koji se koristi. Delegacijski objekt čuva referencu na drugi objekt (*the delegate*) i u određeno vrijeme tom objektu šalje poruku. Poruka informira objekt da se dogodio neki događaj (event). Primjerice u aplikaciji "BackWay" koristim lokacijski delegate koji javlja da je došlo do promjene položaja, tj. lokacije uređaja i to javljanje mi omogućuje da osvježim stanje aplikacije (prikaz smjera, udaljenost itd.). U iOS developmentu se ovaj pattern često koristi.

Kodiranje

Pisanje koda je tema koja je jednostavna, ali opet i komplicirana. Svaka osoba ima različito predznanje i piše različit kod. Neki vole imati više manjih klasa, neki vole imati robusnije klase koje rade više funkcija itd.

Dio razvojnog procesa gdje se piše kod je meni najmanje zanimljiv, ali je odličan jer tek kad se implementira neko rješenje može se vidjeti stvaran rezultat u obliku aplikacije koja se pokrene u simulatoru ili na uređaju. U ovoj fazi također dolazi i do neizbježnih bugova (za koje je ponekad jako teško naći rješenje). Kad se rade neke izmjene češće se rade izmjene u samom kodu, a rjeđe se ide na početak i rade izmjene cijelog dizajna i strukture aplikacije.

Nove verzije

Nakon što se objavi prva verzija aplikacije i nakon što korisnici počnu koristiti aplikaciju, možemo vidjeti da li smo napravili neke greške koje nismo sami primjetili, da li je potrebno neke funkcionalnosti maknuti, neke nove dodati itd. Najbolje je kad su korisnici odmah zadovoljni i kad u aplikaciji nema bugova. U jednoj od verzija BackWay aplikacije dogodio mi se jedan katastrofalan bug zbog kojeg se aplikacija rušila. Taj bug se događao jer nisam pri dodavanju jedne funkcionalnosti koja je bila dostupna u novoj verziji iOS-a provjerio u kodu da li ta funkcionalnost postoji u trenutnom OS-u. S obzirom da sam stavio da podržavam stariju verziju iOS-a, a implementirao sam neke funkcionalnosti novog iOS-a, trebao sam određenu metodu pozivati samo ako je dostupna, a ne bez provjere što je dovelo do rušenje aplikacije i loših recenzija te verzije aplikacije. Sva sreća Apple ima opciju *expedited review* koja omogućuje da primjerice zbog takvih problema (rušenje aplikacije) u roku od 24h možete poslati novu verziju na review u Apple i ispraviti taj problem. Inače bi čekali neko vrijeme u redu za review, a ovako vas ubace preko reda. Tu opciju se ne može koristiti neograničeno i svaki puta kad ju se zatraži potrebno je detaljno objasniti razloge i onda je zahtjev odobren ili ne.

Kao kod svakog softvera tako i kod novih verzija iOS aplikacija neke nove verzije će se raditi zbog bugova, neke zbog novih funkcionalnosti, a neke će donositi velike promjene kao što je potpuni redizajn korisničkog sučelja.

Nepisano je pravilo da je dobro često raditi nove verzije aplikacije, s malim promjenama, jer tako će korisnici imati dojam da se konstantno radi na aplikaciji i da se brine za stanje aplikacije. Razni servisi koji analiziraju stanje aplikacija u App Storeu primjetili su da developeri koji češće osvježavaju verzije aplikacija imaju više skidanja aplikacija od ostalih.

4.4 iOS arhitektura i mogućnosti

Pri razvoju iOS aplikacija koristi se iOS SDK koji sadrži resurse pomoću kojih se razvijaju, pokreću i testiraju aplikacije. Arhitektura iOS-a je slojevita i iOS služi kao posrednik između hardvera i aplikacija. Aplikacije ne komuniciraju direktno s hardverom nego koriste sučelja sustava.

Postoje četiri (4) osnovna sloja iOS-a (poredani od više razine do niže):

- Cocoa Touch
- Media
- Core Services
- Core OS

Frameworks

Tehnologije u iOS-u zapakirane su u razne frameworke. Svaki framework sadrži resurse potrebne za realizaciju nekih zadataka ili operacija i pomoću javnih sučelja se koriste. OS X i iOS dijele dosta istih frameworka, tako da je iOS developerima lako raditi aplikacije za OS X i obratno.

Neki od frameworka:

- UIKit
- Core Graphics
- Core Animation
- Game Kit
- Sprite Kit

- Game Controller
- Core Data
- Foundation
- AV Foundation

Cocoa Touch sloj

Ovaj sloj sadrži ključne frameworke za izgradnju iOS aplikacija. Oni definiraju izgled aplikacije, pružaju osnovnu infrastrukturu aplikacije i osnovne tehnologije kao što su multitasking, unos podataka dodirom, push notifikacije itd. Pri dizajniranju aplikacije važno je proučiti sve te tehnologije kako bi se vidjelo koje tehnologije najbolje ispunjuju potrebe.

Mogao bi svaki dio Cocoa Touch sloja detaljno objasniti, ali radije bih ih nabrojao (možda koji slučajno zaboravim) jer bi mogao u nedogled o svima njima.

- Airdrop - razmjena slika, dokumenata itd. s uređajima blizu nas.
- Text Kit - skup klasa za upravljanje tekstom i korištenje tipografije.
- UIKit Dynamics
- Multitasking
- Auto Layout
- Storyboards
- UI State Preservation
- Apple Push Notification Service
- Local Notifications
- Gesture Recognizers
- Standard System View Controllers

Neki od frameworka u Cocoa Touch:

- Address Book UI
- Event Kit UI

- Game Kit
- iAd
- Map Kit
- Message UI
- Twitter
- UIKit

Media sloj

Ovaj sloj sadrži audio, video i grafičke tehnologije koje se koristi za implementaciju multimedijskog sadržaja u aplikacijama.

Grafičke tehnologije:

- UIKit graphics
- Core Graphics
- Core Animation
- Core Image
- OpenGL ES i GLKit
- Text Kit i Core Text
- Image I/O
- Assets Library

Audio tehnologije:

- Media Player
- AV Foundation
- OpenAL
- Core Audio

Video tehnologije:

- UIImagePickerController
- Media Player
- AV Foundation
- Core Media

Core Services sloj

Ovaj sloj sadrži fundamentalne sustavne servise za aplikacije. Neki od tehnologija koje ovaj sloj sadrži su iCloud, lokacijske tehnologije, društvene mreže, networking itd.

Funkcionalnosti više razine:

- Peer-to-Peer Services
- iCloud Storage
- Automatic Reference Counting
- Block Objects
- Data Protection
- File-Sharing Support
- Grand Central Dispatch
- In-App Purchase
- SQLite
- XML

Neki od frameworka:

- Accounts
- Address Book
- Ad Support
- CFNetwork
- Core Data

- Core Foundation
- Core Location
- Core Media
- Core Motion
- Core Telephony
- Event Kit
- Foundation
- JavaScript Core
- Mobile Core Services
- Multipeer Connectivity
- Newsstand Kit
- Pass Kit
- Quick Look
- Safari Services
- Social
- System Configuration

Core OS sloj

Najniži sloj prema hardveru. Osobno najmanje koristim ovaj sloj, ali to ne znači da ne sadrži korisne stvari, jednostavno nisam u većini slučajeva trebao koristiti niže slojeve iOS-a.

Sadrži:

- Accelerate
- Core Bluetooth
- External Accessory
- Generic Security Services

- Security
- System
- 64-bit Support

4.5 Razvojna okolina

Centar razvoja iOS aplikacija je Xcode IDE. Ostatak razvojne okoline čine ostali alati koji se koriste za razvoj softvera (papir, olovka, softver za obradu fotografija itd.) direktno ili indirektno.

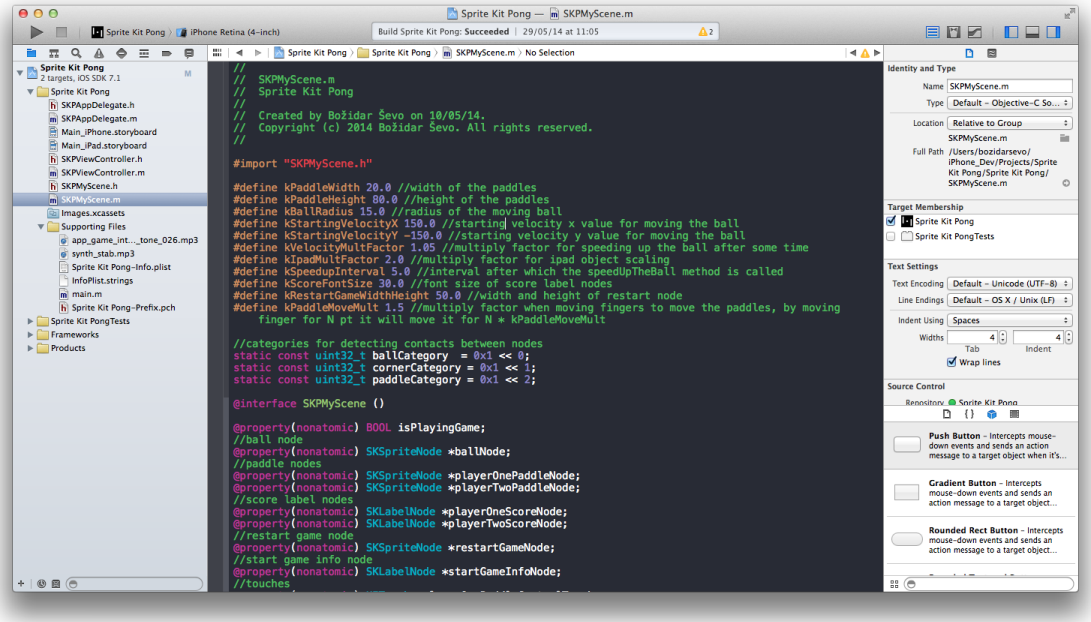
Xcode

Xcode je IDE za razvoj iOS aplikacija. Kao i kod svega kod Applea tako je i slučaj kod razvoja aplikacija, a to je da je sve odlično integrirano. Za razvoj se koristi Apple hardver i Apple softver što donosi odličan tok i užitak u radu na aplikacijama. Xcode sadrži sve potrebne alate i resurse za razvoj iOS aplikacija. Za neke detaljnije stvari će se možda trebati koristiti druge alate, ali za većinu aplikacija Xcode je jedini potreban. Na slici 4.1 se može vidjeti kako izgleda Xcode s otvorenim projektom "Sprite Kit Pong".

Xcode komponente:

- Assistant Editor - dijeli Xcode editor u dva editora, jedan za primaran rad, drugi automatski prikazuje dokumente koji su korisni za rad na primarnom dokumentu.
- Jump Bar - za brzi skok do bilo kojeg resursa u projektu.
- Interface Builder - integriran u Xcode, omogućuje slaganje korisničkog sučelja bez jedne linije koda.
- Version Editor and Source Control
- Testing
- Schemes
- OpenGL Frame Capture
- Documentation
- iOS Simulator
- Compilers

- Graphical Debugger
- Instruments
- Command Line Tools
- Script Languages
- UNIX tools
- i još mnogo toga...



Slika 4.1: primjer izgleda Xcode sučelja

Jedna od najčešće korištenih Xcode komponenti je iOS simulator koji omogućuje da se aplikacije testiraju i bez samog uređaja (iako je najbolje testirati na stvarnom uređaju). Simulator omogućuje da se brzo testiraju aplikacije u raznim verzijama iOS-a ili na različitim tipovima uređaja. Još jedna od odličnih Xcode komponenti, koje neki malo koriste, je "Instruments"[3]. Instruments je odličan alat za poboljšavanje koda, performansi aplikacije, traženja bugova, leakova itd.

[3] `http://developer.apple.com/library/mac/documentation/developertools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html`

Poglavlje 5

iOS Developer Program

Jedan dio samog iOS developmenta je sam razvoj iOS aplikacije, drugi dio je objava aplikacije na App Storeu. Ukoliko se želi aplikacije objaviti u App Storeu potrebno je registrirati se kao developer i platiti iOS Developer Program licencu, koja između ostalog omogućuje distribuciju aplikacija na App Storeu. Također je potrebno biti dio iOS developer programa kako bi se testirale aplikacije na samim uređajima, a ne samo u simulatoru. Potpuno besplatno je cijelo razvojno okruženje i dokumentacija, ali ako se aplikacije žele testirati i objaviti na App Storeu potrebno je platiti godišnju licencu od 99 američkih dolara. Također kao registrirani (s licencom) developer imate pristup raznim Apple developer forumima. Više o iOS developer programu može se naći na - `http://developer.apple.com`.

5.1 Objava na App Store - distribucija

Proces objave aplikacije na App Storeu je jednostavan, ali treba se pridržavati nekih pravila. Objavu na App Storeu najbolje je promatrati kroz cijeli proces razvoja aplikacije. Cijeli razvoj od početka pa sve do objave aplikacije najbolje je podijeliti u nekoliko faza:

1. Uključivanje u iOS Developer Program
2. Razvoj (programiranje, dizajn itd.) i testiranje aplikacije
3. Dodavanje aplikacije u iTunes Connect sustav, uređivanje opisa, screenshotova itd.
4. Slanje aplikacije na Appleu na pregled. Ukoliko je aplikacija odbijena vraćamo se na točku 2.
5. Puštanje aplikacije u App Store

Apple Developer Program

Apple Developer programi pa tako i iOS Developer Program nude razne tehničke resurse, podršku, pristup beta verzijama softvera i općenito sve potrebno kako biste napravili odlične aplikacije za iOS i objavili ih u App Storeu.

U program se možete uključiti kao pojedinac ili kao tvrtka. Prilikom uključivanja u program potrebno je ispuniti neke podatke, osnovne informacije o vama ili o tvrtki. Nakon što se sve to ispuni, prihvate *licence agreements*, plati program, primaju se detalji kako aktivirati članstvo. Nakon što istekne godina dana članstva potrebno je samo produžiti program i ako su se neke informacije o vama promijenile upisati te promjene.

Ako se uključujete kao tvrtka možete dodati više osoba u tim i odrediti kakve privilegije i mogućnosti imaju. Svi članovi moraju biti *Registered Apple Developers*, što mogu besplatno napraviti na `http://developer.apple.com`. Tako neki članovi imaju pravo slanja aplikacije na odobravanje Appleu dok neki mogu samo programirati.

Kao developer se može biti član više timova. Ja sam osobno učlanjen kao pojedinac, ali ako bih radio za neku tvrtku mogao bih biti dodan kao njihov član tima te raditi i svoje i njihove aplikacije.

Apple IDE Xcode nudi dosta opcija što se tiče samog podešavanja okruženja za razvoj i distribuciju aplikacija tako da se može direktno logirati u Xcode s Apple ID-om vezanim uz Developer Program i Xcode će povući sve potrebne informacije o vama za razvoj i distribuciju.

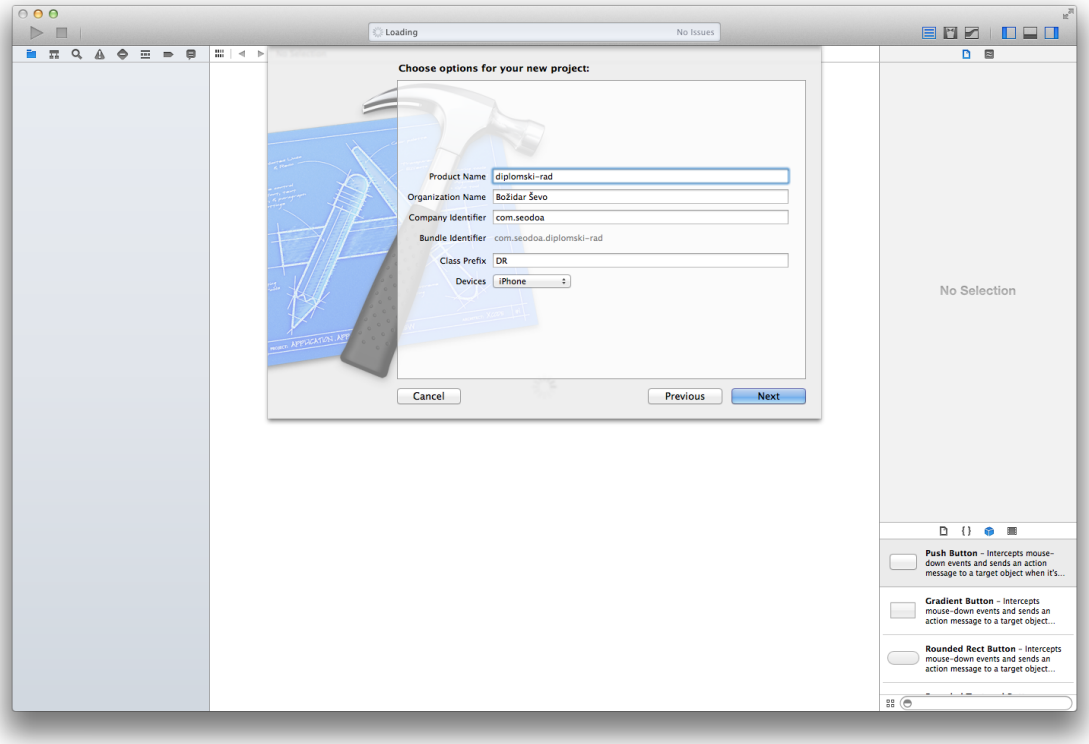
Xcode i distribucija

Kako bi se aplikacije testirale na uređajima i distribuirale u App Store potrebno je podesiti razne certifikate, identifikatore i profile. To se može sve raditi u *iOS Dev Center* - `http://developer.apple.com/devcenter/ios/`, ali se može i unutar samog Xcodea što pojednostavljuje sam proces razvoja jer se ne treba podešavati na više mjesta.

Već kod kreiranja novog projekta u Xcodeu se mogu unijeti informacije o aplikaciji koje će biti potrebne za distribuciju. Naravno, te informacije se ne moraju unijeti odmah, mogu se i kasnije. Neke od informacija koje se upisuju, a mogu se i vidjeti na slici 5.1:

- **Product Name** - ime aplikacije koje će ljudi vidjeti kad skinu aplikaciju
- **Organization Name** - ime koje se koristi kao opis organizacije, bit će dodano u header datoteke u copyright dio
- **Company Identifier** - Product Name i Company Identifier zajedno daju bundle ID koristeći obrnutu DNS notaciju. Bundle ID treba biti jedinstven za aplikaciju što znači da i company identifier mora biti jedinstven. Moj je `com.seodoa`

- **Class Prefix** - može se podesiti automatski prefiks za svaku novo kreiranu klasu
- **Devices** - može se kreirati iPhone, iPad ili univerzalnu aplikaciju (iPhone i iPad)



Slika 5.1: upisivanje podataka o projektu u Xcode

Ikone aplikacije i *launch images* (grafike koje se prikazuju prilikom pokretanja aplikacije) su također bitan dio aplikacije, a dodaju se u Xcodeu. Prilikom pokretanja aplikacije prvo se pokazuje launch grafika dok pokretanje aplikacije ne završi kako bi se pružilo što bolje korisničko iskustvo. Operacijski sustav ikonu aplikacije koristi na više mjesta, na samom *home screenu* s ostalim aplikacijama za pokretanje aplikacije, u pretraživanju uređaja koristeći *spotlight*, postavkama uređaja itd. Ovisno o tome za koje uređaje se razvija potrebno je i imati ikone adekvatnih dimenzija, a to vrijedi i za launch grafike.

Pokretanje na uređaju

Dio razvoja iOS aplikacija je i testiranje aplikacija. Aplikacije se mogu pokretati u iOS simulatoru, ali uvijek je bolje testirati aplikacije na pravom uređaju, posebno ako se podržavaju starije i nove verzije iOS-a kako bismo bili u potpunosti sigurni da će sve raditi kako smo zamislili. Za testiranje na iOS uređaju potrebno je imati/platiti godišnju licencu (ona ista licenca od 99 američkih dolara prije spomenuta).

Slanje aplikacije na review

Slanje prve ili nove verzije aplikacije na review je jedan od "the" trenutaka u razvoju aplikacije, posebno slanje prve verzije. Meni je osobno slanje prve verzije prve aplikacije (SmBuS) imalo nekoliko problema, ali s kasnijim verzijama i aplikacijama sve je išlo glatko. Kad se aplikacija šalje na review u Apple (koristeći Xcode) aplikacija čeka u redu da ju review tim pregleda i odluči da li će biti dostupna za download ili ne. Ukoliko je aplikacija odobrena (ili nova verzija) aplikacija postaje dostupna za skidanje (može se i odrediti datum dostupnosti). Ukoliko aplikacija nije odobrena, review tim javlja razlog(e) neodobravanja i mogu se postavljati pitanja vezana za review proces direktno osobi koja je pregledavala aplikaciju. Nakon toga aplikacija se ponovno šalje na review, s ispravljenim problemima.

Objava aplikacije

Prilikom dodavanja prve ili nove verzije aplikacije u iTunes Connect sučelju može se odrediti datum objave aplikacije (ako bude odobrena). Tako da se može podesiti da je aplikacija dostupna čim bude odobrena ili odabrati neki konkretan datum. Također se može određivati i mijenjati cijene skidanja aplikacije. Dok je aplikacija dostupna za skidanje može se mijenjati cijena skidanja aplikacije za određeni vremenski period. Tu mogućnost mnogi koriste za sniženja i promocije kako bi smanjili cijenu skidanja aplikacije ili stavili da je aplikacija besplatna za skidanje. Nakon objave aplikacije posao nije završen, barem kod ozbiljnih developera. Tada se kreće s praćenjem reakcija korisnika, radom na novim verzijama aplikacije, promociji itd.

5.2 Worldwide Developers Conference - WWDC

Apple Worldwide Developers Conference[1] (WWDC) je konferencija koju Apple održava jednom godišnje u Moscone West, San Francisco, California. WWDC je centralni godišnji

[1] `http://developer.apple.com/wwdc/`

događaj na kojem Apple uobičajeno predstavlja novi hardver i/ili softver i kroz tjedan odvijanja konferencije posjetiteljima (većinom developerima) prezentira novosti. Uobičajeno WWDC počinje s keynote prezentacijom gdje CEO (prije Steve Jobs, danas Tim Cook) otvara konferenciju, iznosi trenutno stanje u Apple svijetu, prezentira novitete (ovaj dio nekad rade i ostali glavni ljudi iz Applea) itd. WWDC je jedna od najpopularnijih konferencija tako da su prijašnjih godina karte od 1599 američkih dolara planule u nekoliko minuta, a čini mi se prošle godine u par sekundi. Ove godine (2014.) uveli su sustav lutrije, tj. svaki developer koji je htio kupiti kartu prijavio se i onda ako bi bio nasumično odabran imao bi pravo kupiti ulaznicu. Otprilike 6000 ljudi ove godine je posjetilo konferenciju.

Apple omogućuje studentima da se prijave za WWDC Student Scholarship (studentska stipendija). Ta stipendija studentima daje besplatnu ulaznicu (ukupno 200 komada), a ove godine su se studenti posebno družili s vrhom Applea (Tim Cook, Jonathan Ive itd.).

Snimke predavanja s konferencije su brzo nakon predavanja dostupne online za pregled i skidanje svim iOS i Mac developerima što omogućuje svima da brzo saznaju sve novosti. Ono što nije dostupno za online su radionice na konferenciji gdje developeri mogu direktno s ljudima iz Applea razgovarati o konkretnim stvarima ili rješavati neke svoje probleme u aplikacijama. Tako primjerice ako imate neki problem s obradom fotografija u aplikaciji možete direktno razgovarati s čovjekom koji je u Appleu odgovoran za framework koji koristite u aplikaciji.

iOS Tech Talks

Dosta ljudi ne uspije otići na WWDC pa Apple ponekad organizira i "iOS Tech Talks"[2], jednodnevne konferencije na kojima prezentiraju novosti i savjete za razvoj iOS aplikacija (koncentracija na novom iOS-u). Zadnji iOS Tech Talks održavali su se u San Franciscu, New Yorku, Tokiju, Šangaju, Berlinu i Londonu. Prvi puta je konferencija bila dvodnevna, tj. održavao se App Developer Day i Game Developer Day. Svaki developer se može prijaviti za maksimalno jedan dan (app ili game i konkretan grad). Imao sam sreću i u 12. mjesecu 2013. godine sam u Berlinu sudjelovao na Game Developer Day. Tech Talks se sastoje od predavanja u glavnoj dvorani i radionica, tj. prostora za direktan razgovor s Apple ljudima u drugoj dvorani. U Berlinu sam upoznao mnoge važne ljude iz Applea (John Geleynse, Allan Schaffer) i upoznao mnoge odlične developere s odličnim aplikacijama i igrama.

[2] `http://developer.apple.com/tech-talks/`

Poglavlje 6

Stanford predavanja

Jedna od prvih, ako ne i prva stanica koju svaka osoba koja kreće u iOS development treba posjetiti. Stanford CS193p kolegij pokriva velik dio iOS developmenta i pokazuje dobar uvid u ono što se na iOS-u može. Naravno, taj kolegij ne pokriva sve, ali je dobar početak za sve one koji se interesiraju za iOS development.

Najjednostavnije je posjetiti njihovu web stranicu[1] i tamo vidjeti kako preuzeti ili pratiti predavanja. A može se i posjetiti iTunes University na webu[2] ili direktno u iTunes[3] i preuzeti snimke predavanja.

Slika 6.1 pokazuje zadnja predavanja koja su dodana na iTunes U.

6.1 Stanford CS193p

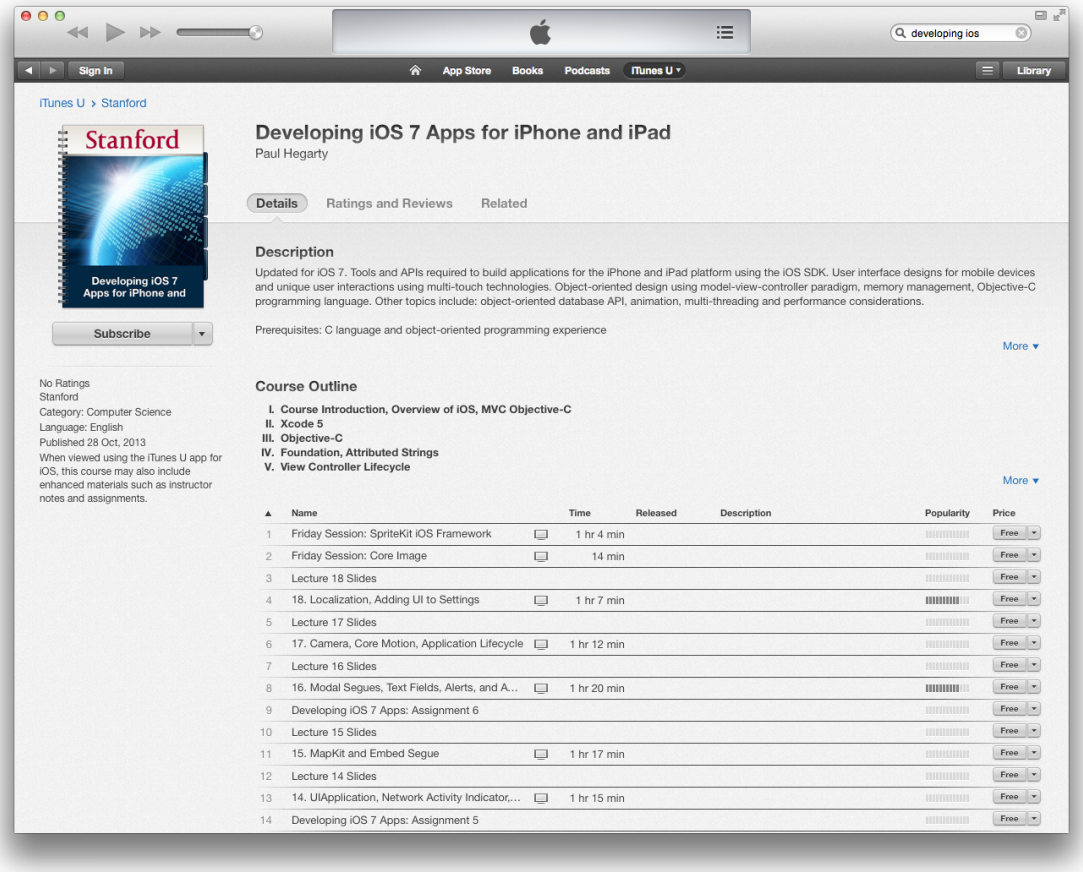
Apple je 2008. godine objavio iPhone SDK, a Stanford University[4] je već 2009. (Slika 6.2) imalo kolegij na kojem se učilo programirati za tu platformu. Još zanimljivija je činjenica da su ta predavanja stavili besplatno na internet kako bi i ostali ljudi imali pristup tim predavanjima. Od 2009. pa do danas taj kolegij se standardno održava i dalje su predavanja dostupna svima online. Osobno ne znam ni jednog iOS developera koji nije pogledao barem jedan semestar tih predavanja.

Teme, lekcije

Popis tema (na engleskom):

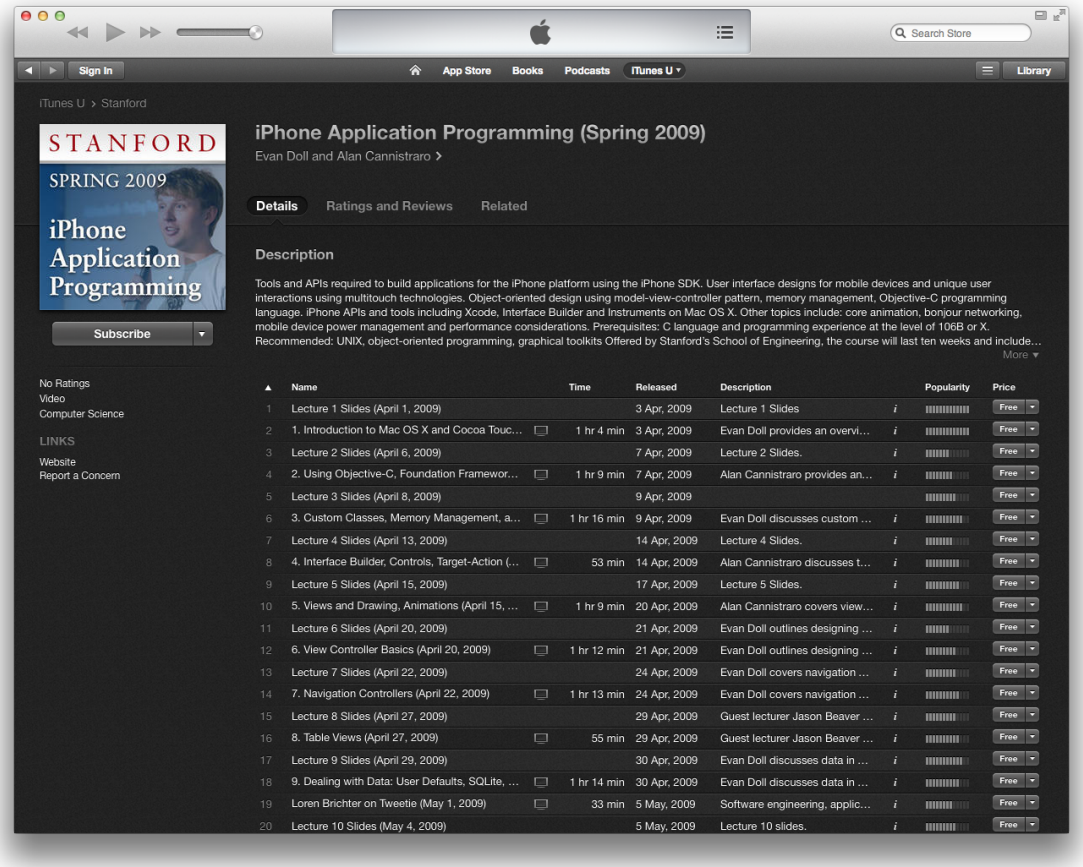
1. *Class Logistics, Overview of iOS, MVC, Objective-C*

[1] `http://cs193p.stanford.edu`
[2] `http://itunes.apple.com/us/course/developing-ios-7-apps-for/id733644550`
[3] `https://www.apple.com/itunes/`
[4] `http://stanford.edu`



Slika 6.1: Stanford CS193p predavanja za iOS7 - 2013/2014, screenshot, 30.6.2014.

2. *Xcode 5*
3. *Objective-C*
4. *Foundation, Attributed Strings*
5. *View Controller Lifecycle*
6. *Polymorphism with Controllers, UINavigation, UITabBar*
7. *Views and Gestures*
8. *Protocols, Blocks, and Animation*



Slika 6.2: Prva Stanford CS193p predavanja za iOS - 2009, screenshot, 30.6.2014.

9. *Animation and Autolayout*
10. *Multithreading, Scroll View*
11. *Table View and iPad*
12. *Documents and Core Data*
13. *Core Data and Table View*
14. *UIApplication, Network Activity, Maps*
15. *MapKit and Embed Segue*

16. *Modal Segues, Text Fields, Alerts and Action Sheets, Camera*
17. *Camera, Core Motion, Application Lifecycle*
18. *Localization, Adding UI to Settings*
19. *Friday sessions:*
    a) *Core Image*
    b) *SpriteKit iOS Framework*

Predavači

Od prve godine tog kolegija pa sve do danas kolegij ima elitne predavače, ili su radili za Apple ili sad rade za Apple, a neki su radili još u NeXT-u s Jobsom. Ono što je najbitnije, odlično su strukturirali kolegij kako bi svatko zainteresiran za iOS development mogao što lakše krenuti i naučiti kako razvijati i programirati za iOS.

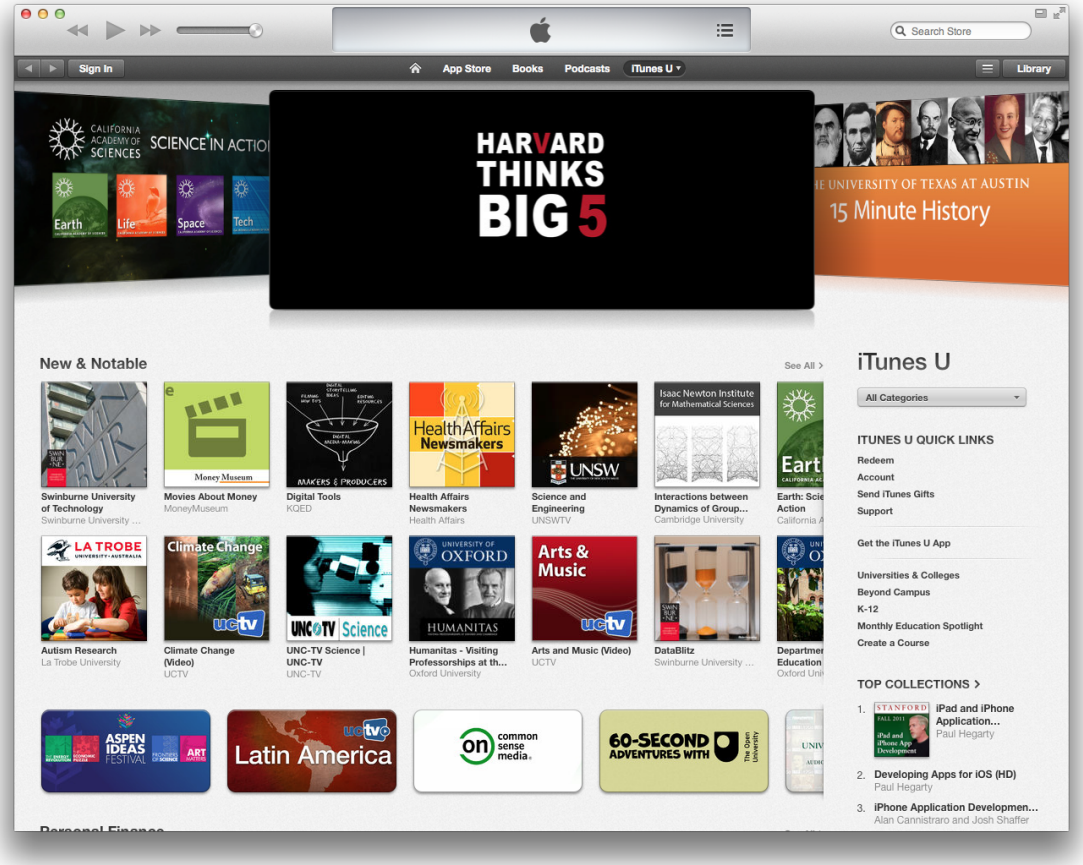
Zasad su ovi predavači bili prisutni:

- Evan Doll[5] - bivši senior iPhone inženjer u Appleu, danas osnivač tvrtke Flipboard[6]; držao predavanja 2009.
- Alan Cannistraro[7] - danas radi u Facebooku (London), prije radio na nekoliko položaja u Appleu; držao predavanja 2009. i 2010.
- Josh Shaffer[8] - radi u Appleu ; držao predavanja 2009. i 2010.
- Paul Hegarty[9] - radio u NeXT; držao predavanja 2010., 2011., 2012. i 2013.

6.2 iTunes University

Danas ima dosta izvora, točnije web stranica gdje se može besplatno ili jako povoljno učiti programiranje. Jedan od tih izvora je i iTunes University[10] od Applea. Tamo mnoga svjetska sveučilišta stavljaju svoja predavanja iz ekonomije, medicine, dizajna, fizike, kemije pa sve do programiranja (npr. za iOS). Stanfordov CS193p je jedan od kolegija koji se nalaze na iTunes U i koliko znam najpopularniji je. Slika 6.3 pokazuje početnu stranicu iTunes U gdje se mogu vidjeti razna zanimljiva predavanja i kolegiji.

[5] `http://www.linkedin.com/in/evandoll`
[6] `http://flipboard.com`
[7] `https://twitter.com/accannis`
[8] `http://twitter.com/joshshaffer`
[9] `http://www.linkedin.com/in/phegarty`
[10] `http://www.apple.com/support/itunes-u/`



Slika 6.3: iTunes University u iTunes, screenshot, 30.6.2014.

Poglavlje 7

Aplikacije

U ovom poglavlju posvetit ću se mojim iskustvima u razvoju iOS aplikacija u zadnjih nešto više od 4 godine od kako sam krenuo u iOS development. Također uz primjere mojih aplikacija pokazat ću neke principe kojih sam se držao i dati neke savjete kako bi razvoj iOS aplikacija bio što ugodniji i produktivniji.

7.1 Moje aplikacije

Popis mojih aplikacija i adekvatne linkove za skidanje može se naći na `http://seodoa.com/apps` ili `http://itunes.apple.com/us/artist/bsevolution/id396541603`. Sve moje aplikacije na App Storeu dosad poredane od najstarije do najnovije s obzirom na izlazak prve verzije:

- SmBuS (21.10.2010.)
- MyPass! (2.8.2011.)
- The Intersect (13.10.2011.)
- BackWay (23.5.2012.)
- Fresh Island Festival (6.7.2013.)
- RockPaperStartups (16.7.2013.)
- Spark.me (20.9.2013.)
- ZABA4U (26.7.2013.)
- ERSTE4U (1.8.2013.)

- PBZ4U (1.8.2013.)
- RBA4U (1.8.2013.)
- ColorTap! (20.9.2013.)
- Color Maze (15.5.2014.)

Za neke aplikacije ću staviti screenshotove, a za neke ne jer se ionako aktualna stanja aplikacija mogu pronaći u App Storeu.

SmBuS

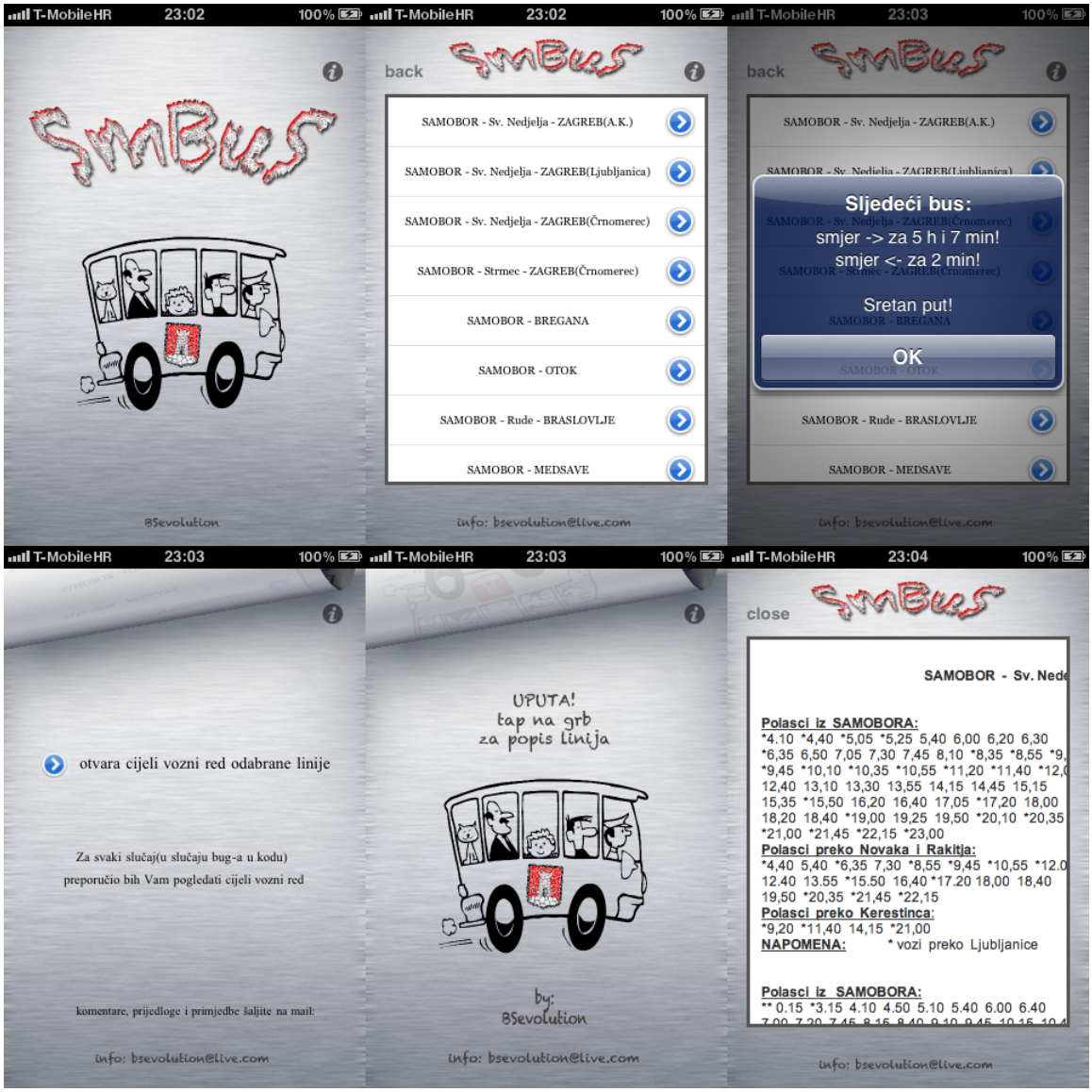
SmBuS je prva aplikacija koju sam izradio i objavio (21.10.2010.) u App Storeu. Kad sam krenuo s učenjem iOS developmenta imao sam nekoliko ideja za aplikacije, ali ni jedna od tih nije bila za SmBuS. Jedan dan sam kod poznanika (Luka Sučić) na Facebooku vidio da ga je jedan prijatelj priupitao da li bi Luka i njegova firma možda mogli napraviti aplikaciju koja bi služila kao vozni red za Samoborček i Autoturist autobuse koji voze u Samoboru i okolici (slika 7.2). Pošto sam taman naučio dovoljno za izradu neke ozbiljnije aplikacije, a i takva aplikacija bi meni bila korisna, rekao sam čovjeku da bih ja rado napravio tu aplikaciju i da ću mu javiti kad bude gotova. To je bilo 24.8.2010. i aplikacija je 21.10.2010. bila dostupna za skidanje u App Storeu.

SmBuS aplikacija korisniku omogućuje da za odabranu autobusnu liniju sazna kad polazi sljedeći autobus (za koliko sati, minuta itd.) i da vidi cijeli vozni red linija. S obzirom da su do tada svi morali stalno ići na web autoprijevoznika ili si spremati screenshotove voznog reda, ova aplikacija je svima bila korisna i potrebna. I najbitnija stvar u aplikaciji je ta da radi u potpunosti offline što je jako bitno jer su glavni korisnici aplikacije učenici i studenti koji tada (a i danas) nisu imali velike internet pakete na svojim uređajima, a ovako su se uvijek mogli pouzdati u aplikaciju.

Kao što se vidi na slici 7.1 prva verzija aplikacije je bila "zanimljivog" dizajna. S obzirom da nisam imao iskustva s ozbiljnim dizajnom i ozbiljnim razvojem softvera bio sam jako zadovoljan s prvom verzijom. Kasnije verzije, pogotovo najnovija (slika 7.3) je ipak na jednoj višoj razini, a i dalje mnogima donosi korisne informacije.

SmBuS je moje "dijete" u smislu razvoja softvera i koliko god jednostavna aplikacija bila, donijela mi je mnogo toga. Osjećaj kad vidite ljude u autobusu kako pričaju o vašoj aplikaciji i preporučuju ju svojim prijateljima/icama je neprocjenjiv.

Broj skidanja: 2385 (do 27.8.2014.)



Slika 7.1: Prva verzija SmBuS aplikacije

MyPass!

Ideja za aplikaciju "MyPass!" mi je nekako "niodkuda" došla. Svi koristimo razne PIN-ove i lozinke pri korištenju bankomata, internet bankarstva, raznih web stranica itd. Postojale su neke aplikacije (mobilne i desktop) koje omogućuju spremanje svih tih lozinka na jedno

Slika 7.2: Prijedlog za SmBuS aplikaciju - 24.8.2010.

mjesto, ali ja sam htio aplikaciju koja radi offline i sprema sve na uređaj radi potpune sigurnosti. Znam da mnogi ljudi (većinom stariji) ne vjeruju servisima koji rade online sinkronizacije i slične stvari tako da sam odlučio napraviti offline verziju takvih aplikacija.

Cilj MyPass aplikacije je spremanje vama važnih PIN-ova, lozinki i ostalih vezanih informacija na način da glavna lozinka otključava prikaz svih tih spremljenih lozinki. Kad se aplikacija otvori korisnik mora upisati ili podesiti (ako prvi put otvara) glavnu lozinku i prikazuju se spremljene lozinke i podaci. Ukoliko se glavna lozinka upiše tri puta krivo svi podaci u aplikaciji se brišu. Ukoliko je uređaj stavljen u *sleep mode* ili se korisnik prebaci u drugu aplikaciju, MyPass se automatski prebaci na početni ekran (zaključava se) i opet će tražiti glavnu lozinku od korisnika. Korisnik može i ručno zaključati aplikaciju ili obrisati sve podatke.

Trenutno je aktivna verzija 2.0 i u ovom videu `http://www.youtube.com/watch?v=aiOiQaMB9l0` se može vidjeti kako otprilike radi. Aplikacija je lokalizirana/prevedena (slika 7.4) na engleski, francuski, njemački, talijanski, španjolski, japanski, kineski i hrvatski.

Broj skidanja: 14427 (do 27.8.2014.)

Slika 7.3: Najnovija verzija (2.x) SmBuS aplikacije

The Intersect

Prije razvoja ove aplikacije brat i ja smo pratili američku seriju “Chuck”. U toj seriji glavni lik Chuck pomoću *Intersecta* dobije sve informacije koje posjeduju tajne službe na način da mu se sav taj sadržaj uploada u mozak. Htio sam napraviti zabavnu verziju tog stroja tako da sam napravio simulaciju stroja iz serije. Kad se otvori aplikacija korisnik mora odgovoriti na neka pitanja vezana za seriju kako bi dobio mogućnost pokretanja Intersecta, a kad pokrene Intersect krenu se prikazivati velikom brzinom razne slike (tako se stroj ponaša i u seriji).

Broj skidanja: 15030 (do 27.8.2014.)

Slika 7.4: Najnovija verzija MyPass! aplikacije - 2.0

BackWay

BackWay je aplikacija koja korisnicima omogućuje da offline spremaju njima važne lokacije (važne informacije o lokaciji, fotografija lokacije itd.). Ukoliko se korisnik želi vratiti do te lokacije BackWay vrlo jednostavno pokazuje smjer i udaljenost do te ciljne lokacije, a da pri tome troši vrlo malo baterije i ne zahtijeva da korisnik ima pristup internetu što je jako korisno ako se primjerice planinari ili putuje u inozemstvo (*internet roaming*). Više o BackWay aplikaciji nešto kasnije.

Broj skidanja: 3575 (do 27.8.2014.)

Fresh Island Festival

Nakon što sam napravio aplikaciju BackWay, na konferenciji Webfest u Budvi vidio sam da bi bilo dobro funkcionalnosti BackWay aplikacije ubaciti u aplikacije za evente pa tako

i konferencije, jer na njih većinom dolaze ljudi van grada ili države pa bi im offline navigacija i informacije jako dobro došli.

Na Twitteru sam vidio da su pokretači Fresh Island Festivala rekli da im se sviđa BackWay pa sam ih kontaktirao. Predložio sam im da im napravim (i napravio sam) jednu takvu aplikaciju koja bi imala funkcionalnosti BackWaya za najbitnije lokacije za vrijeme tog festivala, a imala bi i sve bitne informacije koje bi posjetiteljima trebale za vrijeme festivala kao što je raspored izvođača, važni telefonski brojevi itd.

Broj skidanja: 247 (do 27.8.2014.)

RockPaperStartups

Moja druga aplikacija za evente, točnije za RockPaperStartups (RPS) konferenciju u Rijeci. Korisnici na raspolaganju imaju raspored konferencije, informacije o govornicima, najvažnije informacije o gradu u kojem se održava konferencija i važne lokacije s BackWay navigacijom.

Kad sam išao na konferenciju napokon sam i osobno upoznao Sanjina Celeskog kojeg sam spomenuo u Uvodu.

Broj skidanja: 123 (do 27.8.2014.)

Spark.me

Spark.me je konferencija u Budvi u Crnoj Gori koja je nasljednik Webfest konferencije koju sam posjetio 2012. i predstavljao BackWay aplikaciju. Za prvu Spark.me konferenciju 2013. godine napravio sam aplikaciju za posjetitelje. Kao i aplikacije za Fresh Island i RPS, Spark.me aplikacija sadrži sve važne informacije potrebne posjetiteljima i sve važne lokacije koje bi trebale posjetiteljima ili ih možda zanimale.

Broj skidanja: 117 (do 27.8.2014.)

ZABA4U

2013. godine sam se po drugi puta prijavio na studentsko natjecanje u izradi mobilnih i web aplikacija "App Start Contest" u organizaciji udruge eStudent. Prijavio sam se s bratom u timu za partnersku temu Zagrebačke banke "Mobilno bankarstvo". Izradili smo aplikaciju ZABA4U i osvojili prvu nagradu od 10 000 kuna.

Aplikacija je bazirana na BackWay aplikaciji i ima spremljene sve poslovnice i bankomate Zagrebačke banke. Korisnik može brzo vidjeti koja poslovnica ili bankomat su najbliži i dobiti BackWay offline navigaciju.

Broj skidanja: 221 (do 27.8.2014.)

Nakon natjecanja odlučio sam napraviti i verzije za ostale bitnije banke u Hrvatskoj tako da sam napravio i ove aplikacije:

- ERSTE4U - za korisnike Erste&Steiermärkische banku.
- PBZ4U - za korisnike Privredne banke Zagreb.
- RBA4U - za korisnike Raiffeisen Bank Austria.

ColorTap!

ColorTap! je moja prva igrica ikad napravljena. Napravio sam ju koristeći tada tek izašli Appleov framework Sprite Kit za izradu igara za iOS i OS X. Igrač treba tapnuti na točno određene geometrijske oblike (krug, trokut, kvadrat) točno određene boje (svaki level ima drukčije elemente) i tako skuplja bodove. Svaki sljedeći level elementi se brže kreću po ekranu i postaju sve manji i manji. Više o igrama pa tako i ColorTap! igri u sljedećem poglavlju.

Broj skidanja: 816 (do 27.8.2014.)

Color Maze

Moja druga igra za iOS s kojom sam se po treći put natjecao na App Start Contest natjecanju. Cilj igre je na 60 jedinstvenih levela u određenom vremenu kuglicu određene boje dovesti do kružnice iste te boje. Igrač koristeći uređaj, tj. gibanjem uređaja kako bi se aktivirale promjene u akcelerometru, pomiče kuglicu kroz labirint. Ukoliko igrač uspije dovesti lopticu do cilja u zadanom vremenu i bez da dodirne kružnice drugih boja, tada je uspješno završio level i osvaja zaslužene bodove i zvjezdice. Više o igrama pa tako i Color Maze igri u sljedećem poglavlju.

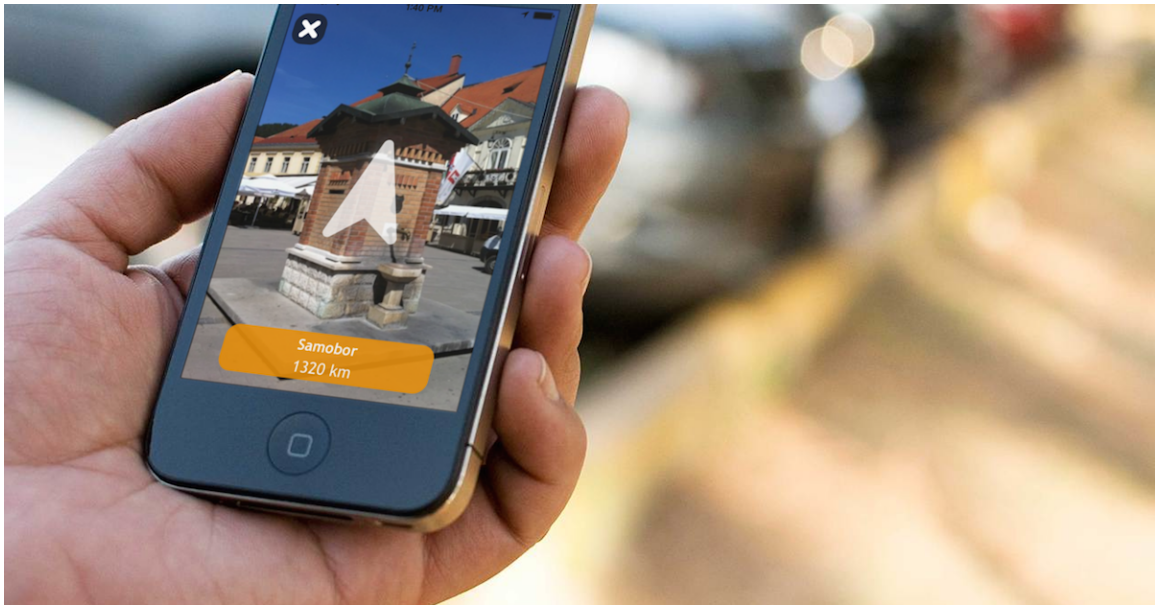
Broj skidanja: 1817 (do 27.8.2014.)

7.2 BackWay

BackWay je jedna od dražih aplikacija koje sam napravio. BackWay je aplikacija koja korisnicima omogućuje da offline spremaju njima važne lokacije (važne informacije o lokaciji, fotografija lokacije itd.). Ukoliko se korisnik želi vratiti do neke lokacije BackWay vrlo jednostavno pokazuje smjer i udaljenost do te ciljne lokacije, a da pri tome troši vrlo malo baterije i ne zahtjeva da korisnik ima pristup internetu što je jako korisno ako se primjerice planinari ili putuje u inozemstvo.

Ideja

Ne sjećam se točno kako sam došao do konkretne ideje za aplikaciju, ali se otprilike sjećam nekih indikacija. Sjećam se samo da mi je palo na pamet da bi bilo dobro imati neku jed-

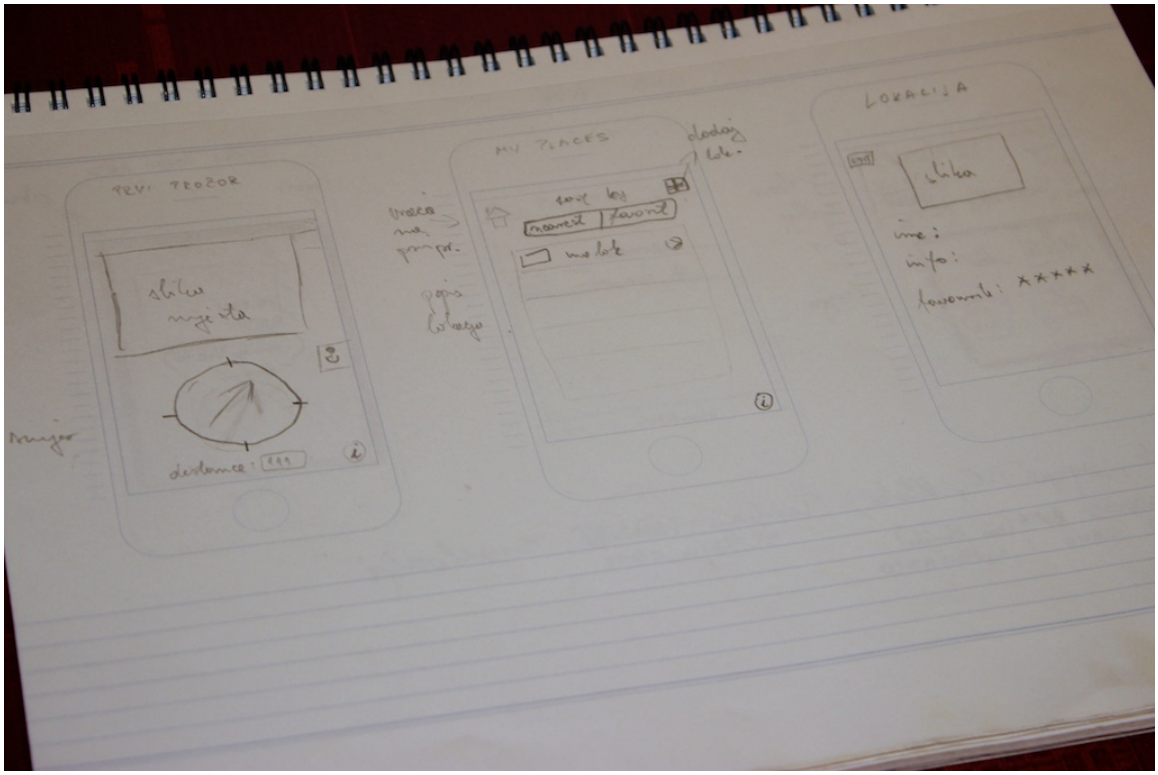


Slika 7.5: Navigacija, tj. pokazivanje smjera i udaljenosti u aplikaciji BackWay

nostavnu aplikaciju koja bi spremala koordinate posjećenih lokacija i omogućavala najjednostavniju navigaciju, a to su smjer i udaljenost (zračna linija). Pretraživao sam App Store kako bi vidio da li ima nekih takvih aplikacija. Bilo ih je, ali su bile jako loše napravljene. Pod jako loše mislim da je dizajn bio katastrofalan, korisničko iskustvo je bilo jako loše i u globalu nisu bile ugodne oku i nisu bile skroz ogoljene kao što bi moja idejna aplikacija bila. Malo sam počeo skicirati kako bi aplikacija izgledala, kako bi funkcionirala (slika 7.6) itd. Taman u to vrijeme kad sam krenuo razmišljati da bi i izradio tu aplikaciju vidio sam da se organizira studentsko natjecanje u izradi aplikacija "App Start Contest" i odlučio sam se prijaviti na to natjecanje sa sestričnom Almom (radila je dizajn aplikacije), više o ASC-u u nastavku.

Funkcionalnosti

Glavne dvije funkcionalnosti BackWay aplikacije su spremanje zanimljivih lokacija i navigacija, tj. vraćanje prema tim lokacijama. Kad korisnik želi spremiti lokaciju na kojoj se trenutno nalazi jednostavno odabere "Add new tag", uređaj automatski uzima koordinate lokacije, korisnik može i ne mora napisati neki opis lokacije, a može i uslikati fotografiju lokacije kako bi se bolje sjetio kasnije koja je to lokacija bila. Kad se lokacija spremi, svi podaci su spremljeni na uređaju kako bi aplikacija u potpunosti bila korisna u offline



Slika 7.6: prve skice BackWay aplikacije

okruženju.

Ukoliko se korisnik želi vratiti nekoj lokaciji odabere neku od spremljenih lokacija i opciju "Guide" koja pokreće navigaciju kao što je na slici 7.5. Tada aplikacija prati promjene u lokaciji uređaja i smjeru u kojem je okrenut uređaj i osvježava sučelje tako da osvježava trenutnu udaljenost od ciljne lokacije i podešava adekvatan smjer prema lokaciji (bijela strelica). Također cijelu pozadinu tog navigacijskog sučelja čini fotografija lokacije.

Pri spremanju lokacije korisnik može upisati naslov (*title*), opis (*description*), odabrati neku kategoriju (koristeći vizualni *category picker* s ikonama za određenu kategoriju) i uslikati fotografiju. Sve te podatke uvijek može promijeniti i spremiti promjene. Core Location je glavni framework koji se koristi kako bi se aplikacija obogatila lokacijskim mogućnostima.

U verziji 1.2 dodao sam i opciju za *share* (dijeljenje) lokacija s drugim ljudima. Postoji offline i online dijeljenje lokacija. Ako se odabere online dijeljenje onda se svi podaci uploadaju na BackWay server i korisnik dobije jedinstven url koji može dalje dijeliti, staviti na Facebook itd. Primjer takvog url-a je `http://backway.me/l/?l=samobor` za pregled

na webu i *url scheme* za import `backway://?id=samobor`. Kad se takav url otvori u browseru, pokazuju se fotografija lokacije, opis lokacije i oznaka na karti. Ako se otvori na iOS uređaju onda se pokaže i jedan gumb koji pokreće import (pomoću url schemes) te lokacije u BackWay aplikaciju ili otvara App Store kako bi korisnik skinuo BackWay aplikaciju. Ukoliko se odabere offline dijeljenje onda se također kreira url za pregled te lokacije na webu `http://backway.me/l/?lo=15.709691&la=45.801864&t=Samobor` i direktan import url za import lokacije u aplikaciju `backway://?lo=15.709691&la=45.801864&t=Samobor`.

U verziji 2.0 dodao sam mogućnost pretraživanja lokacija iz popularnog servisa Foursquare[1]. Korisnik može primjerice tražiti "Apple Store, San Francisco" i dobit će popis Apple Storeova u San Franciscu ili blizu San Francisca. Te lokacije koje dobije kao rezultat korisnik može importati (spremiti) u aplikaciju kako bi ih imao na raspolaganju za offline korištenje, tj. navigaciju prema tim lokacijama.

Sve funkcionalnosti aplikacije uz demonstraciju najbolje je pogledati na Youtube kanalu: `http://www.youtube.com/user/backwayapp/videos`.

App Start Contest

S obzirom da sam BackWay aplikaciju razvijao u sklopu natjecanja App Start Contest[2] htio bih iznijeti svoja iskustva s natjecanjem App Start Contest, neke savjete budućim natjecanjima i razloge zašto bi se netko uopće uključio u takva natjecanja.

ASC je edukativno studentsko natjecanje u izradi mobilnih i web aplikacija koje organizira studentska udruga "eStudent". Kao što sami organizatori kažu cilj natjecanja je potaknuti studente na učenje, razvijanje programskih rješenja, povezati uspješne i talentirane studente s poduzećima i omogućiti studentima da svoje projekte predstave široj masi (klijenti, investitori, mediji itd.). Dio natjecanja su i edukativna predavanja na kojima razna poduzeća govore o tehničkim, poslovnim i raznim ostalim temama koje su korisne za razvoj projekata. Studenti koji se prijave na natjecanje, pohađaju predavanja i predaju završno rješenje mogu dobiti i 4 ECTS boda jer je App Start Contest službena vještina na Fakultetu elektrotehnike i računarstva.

S obzirom da naši fakulteti ne prate aktualna stanja u ICT-u, App Start Contest je odlična stvar koja omogućuje da se već aktivni studenti pokažu široj masi, a i kako bi se onima manje aktivnima pokazalo što se sve može s mobilnim platformama i webom. Po meni bi svaki fakultet vezan za ICT (FER, PMF, FOI itd.) trebao imati što više kolegija u kojima se izrađuju konkretni projekti, a ne da se radi samo teorija i neki neprimjenjivi zadaci (što se tiče programiranja i razvoja softvera). ASC je odlično mjesto za povezivanje s potencijalnim budućim poslodavcima, kolegama ili poslovnim partnerima.

[1] `http://foursquare.com`

[2] `http://www.estudent.hr/app-start-contest/asc-vijesti`

Ono što je najvažnije je da svaki student koji studira nešto vezano za ICT nema što izgubiti natjecanjem na ASC-u i neprijavljivanjem si samo smanjuje šansu za dobrim zaposlenjem u budućnosti (ovdje ne mislim na one studente koji se nevezano za ASC bave svojim projektima i već imaju sigurno zaposlenje ako ne nastave sa svojim projektima). Ja sam siguran da će se svaki student koji uz fakultet radi na svojim projektima ili je općenito uključen u ICT zajednicu u "najgorem" slučaju nakon fakulteta zaposliti u nekom poduzeću. Developeri mobilnih aplikacija su vrlo traženi, tako primjerice ja u ovom trenutku pisanja imam nekoliko ponuda za zaposlenje, većinom domaćih poduzeća uz nekoliko stranih, bez aktivnog traženja zaposlenja. Tako da svim studentima savjetujem da ne kukaju kako je sve loše i neka se pokrenu jer danas uz internet i jednostavnu povezanost s cijelim svijetom mali čovjek može napraviti čuda.

7.3 Konferencije i eventi

Posjetio sam razne stručne i nestručne konferencije i evente koji se detaljnije mogu vidjeti u životopisu kasnije. Neki bitniji eventi su:

- Webcamp konferencija (2014.) - volonter za foto, video i ostale standardne stvari u pripremi lokacije (hrana, piće)
- Apps World London 2013, Indie Game Zone - izlagač s igrom ColorTap!
- natjecatelj na 3 App Start Contesta - 2014., 2013. i 2012. godine
- mentor na iOS radionici (2013.)
- Shift konferencija (2013.)
- finalist Business Plan Contesta (2013.)
- jedini hrvatski predstavnik u Londonu na "European Space Solutions conference" (2012.)
- Startup Camp Vis 2. nagrada (2012.)
- Case Study Competition finalist (2011.)

Poglavlje 8

Igre

U ovom poglavlju bavit ću se malo detaljnije igrama koje sam napravio za iOS, koristeći Sprite Kit. Također ću iznijeti neke zanimljivosti i mogućnosti koje Sprite Kit pruža iskusnim, a i onima manje iskusnim developerima. Prije nego je izašao Sprite Kit (WWDC 2013.) planirao sam proučiti Cocos2D (2D game framework), ali nikako nisam stigao uz fakultet, ostali iOS development itd. Kad je Sprite Kit objavljen i kad sam vidio predavanja s WWDC-a o Sprite Kitu, znao sam da trebam baciti oko i krenuti proučavati što se sve može.

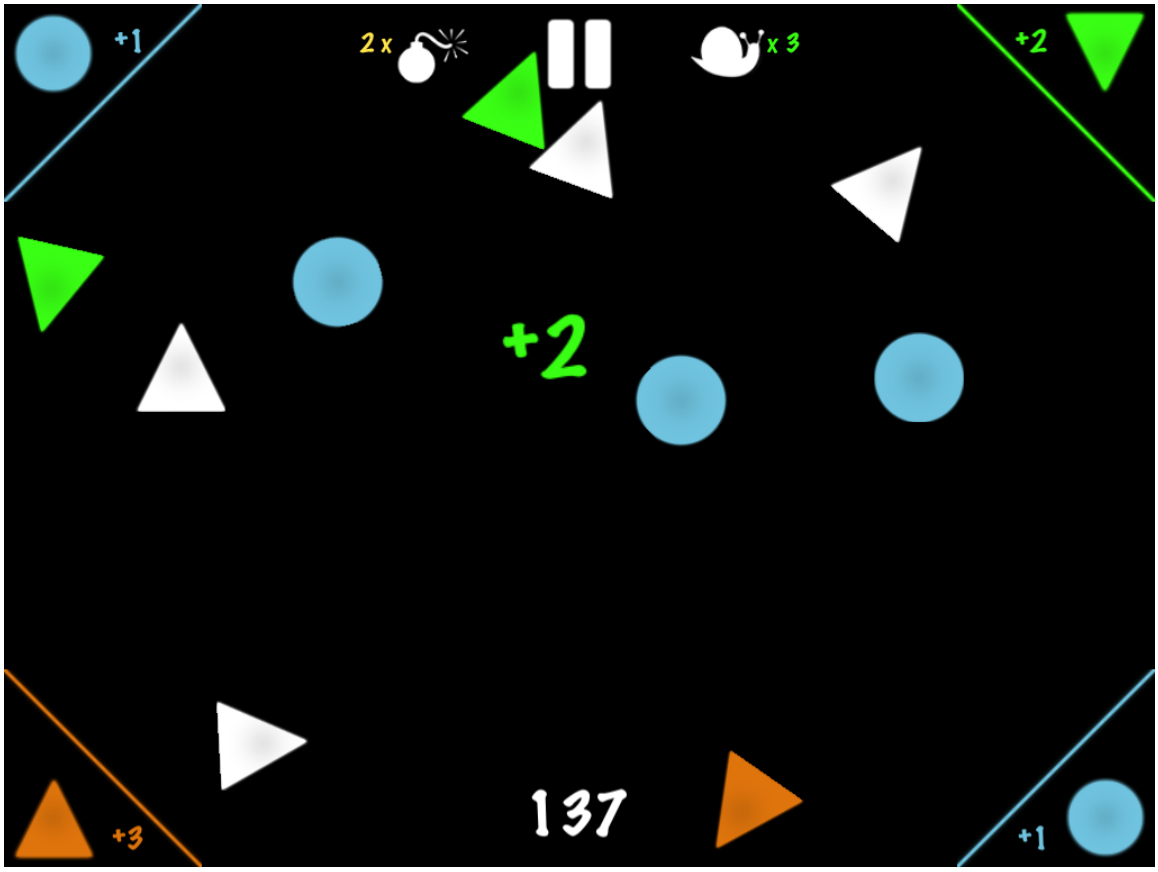
U nastavku ovog poglavlja glavne teme su moje igre "ColorTap!" i "Color Maze", razvoj igara za iOS pomoću Sprite Kita i moji *open source* projekti.

8.1 ColorTap!

ColorTap! je moja prva igra ikad napravljena. Igra ima 13 levela, a na svakom levelu igrač treba tapnuti na točno određene geometrijske oblike (krug, trokut, kvadrat) točno određene boje (svaki level ima drukčije elemente) i tako skuplja bodove. Svaki sljedeći level elementi se brže kreću po ekranu i postaju sve manji i manji. Na svakom levelu se u kutevima nalazi informacija koje elemente bi trebali tapnuti kako bi igrač znao koji su "pravi", a koji "krivi" elementi (slika 8.1). Ukoliko je igrač dosta brz može dobiti i dvostruke bodove za određeni element. Također postoje i opcije "destroy all" za automatsko uništavanje svih elemenata na levelu i "slow down" za usporavanje svih elemenata na levelu. Te opcije igrač može kupiti kroz *In-App Purchase*[1] unutar igre.

Igra je univerzalna, što znači da je napravljena za sve iOS uređaje, iPhone, iPad i iPod touch. Skidanjem jedne verzije igra se može igrati na svim tim uređajima.

[1] `https://developer.apple.com/in-app-purchase/`



Slika 8.1: Primjer jednog levela u ColorTap!

Ideja

Većina igara koje imam na svojim iOS uređajima (iPhone 4 i iPad mini) su jednostavne igre koje se igraju s malim brojem gesti, tapkanja itd. Kao i svaka ideja i ideja za ColorTap! nastala je pod utjecajem raznih igara koje sam igrao do tada. Čini mi se da je najveći utjecaj imala igra "Fruit Ninja"[2] u kojoj igrač treba prstom presjeći voće koje leti po ekranu. S obzirom da volim jednostavne igre i grafički "ogoljene" igre palo mi je na pamet da bih mogao napraviti igru gdje po ekranu lete geometrijski oblici raznih boja i da igrač mora brzo reagirati i tapnuti na točne elemente. Ono što sam kasnije primijetio (posebno dok sam bio u Londonu kao izlagač na Apps World) jest da je igra jako korisna za djecu.

Dakle cilj igre je tapnuti pravi element prave boje. Glavne riječi koje su bit igre su

[2] `http://fruitninja.com`

color i *tap*. Kad sam razmišljao o imenu igre prvo sam ju mislio nazvati "TouchACol", ali nakon nekog vremena razmišljanja mama mi je predložila ime "ColorTap" i to ime je pobijedilo. S obzirom da je u App Storeu postojala aplikacija imena ColorTap, imenu sam dodao samo uskličnik i to je bilo konačno ime igre "ColorTap!".

Od papira i olovke do igre u App Storeu u 60 dana

Sprite Kit je objavljen na WWDC 2013. godine. S obzirom da sam iOS developer, imao sam pristup snimkama predavanja s konferencije. Čim su snimke bile objavljene pregledao sam ih. S obzirom da sam već neko vrijeme planirao krenuti malo i u razvoj igara, odlučio sam se baciti na Sprite Kit. Činjenica da je Sprite Kit Appleov framework daje sigurnost da će se framework održavati i da će biti optimiziran za iOS uređaje (što ponekad nije slučaj s ostalima). Na WWDC-u je također bio predstavljen iOS 7 za koji je rečeno da će izaći (zajedno sa Sprite Kitom) na jesen 2013. Znao sam da je to prilika da preko ljeta (dok mi developeri imamo pristup beta verzijama SDK i iOS-a) napravim igru sa Sprite Kitom i da budem među prvima koji će napraviti igru sa Sprite Kitom. Mislim da je ColorTap! bila prva ili druga igra na App Storeu napravljena sa Sprite Kitom.

Sve aplikacije koje sam radio skiciram u bilježnici "App Sketchbook" koja na svakoj stranici ima otisnut iPhone. Tako sam krenuo i sa skicama i idejama za igru ColorTap! (slika 8.2).

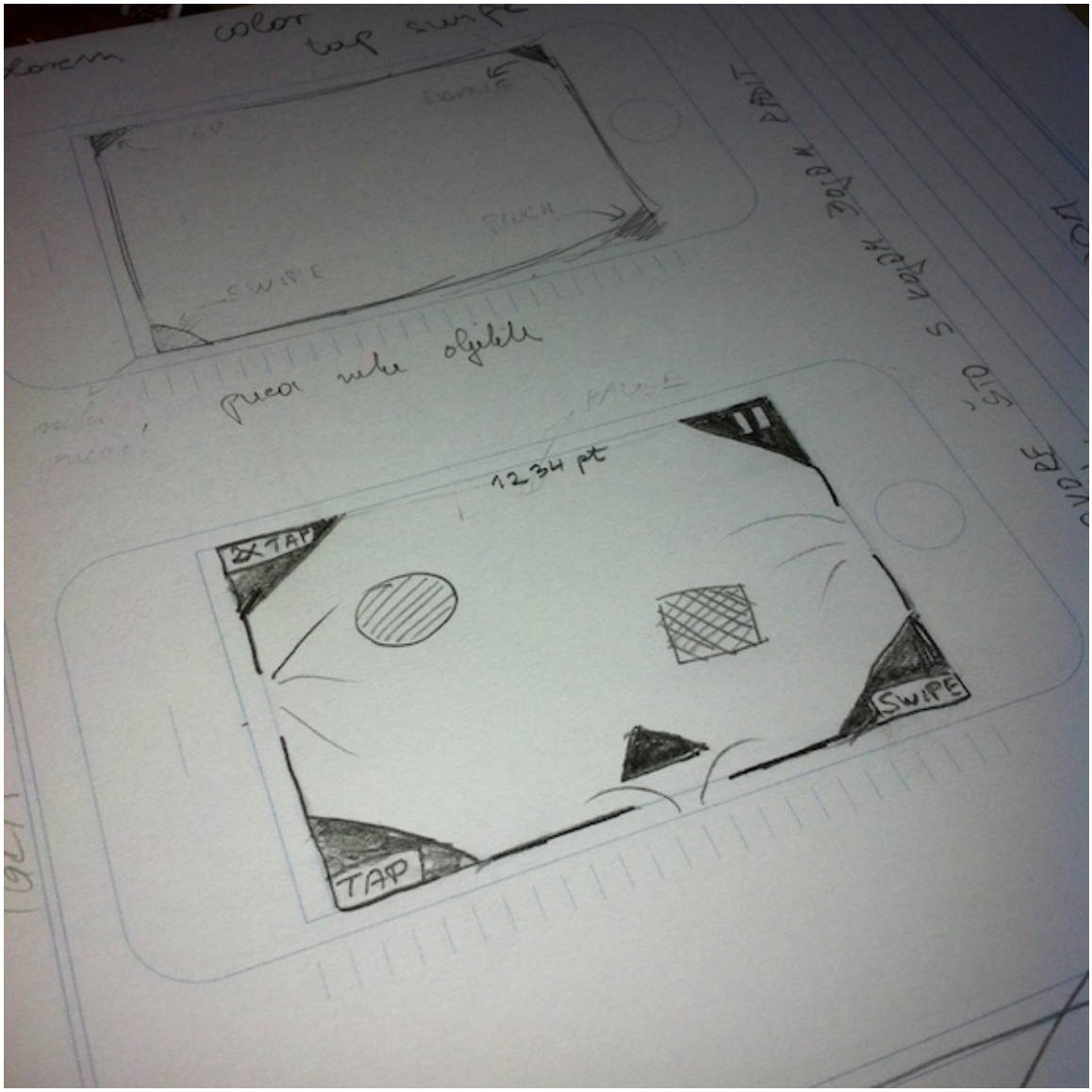
S obzirom da je Sprite Kit bio tada nov krenuo sam sa "Sprite Kit Programming Guide"[3] koji je glavni vodič za razvoj igara sa Sprite Kitom. Taj vodič sadrži sve glavne smjernice i najbolje prakse pri razvoju igara sa Sprite Kitom. Nakon što sam prošao taj vodič krenuo sam polako raditi glavne dijelove buduće igre. Prvi cilj je bio napraviti glavni dio gameplaya, a to je detekcija tapkanja pravog ili krivog elemenata. Nakon toga slijedio je rad na kosturu sučelja igre (scene), levelima, glazbi, testiranju i radu na detaljima. S obzirom da mi je to bila prva igra cijeli taj proces je bio učenje i vježbanje.

Nakon što je igra objavljena u App Storeu odlučio sam tu cijelu priču o razvoju igre u kratkom periodu objaviti online pa sam napisao jedan zanimljiv blog post na mojoj web stranici naslova "How I made my first game in 60 days with Sprite Kit" - `http://seodoa.com/2013/10/31/how-i-made-my-first-game-in-60-days-with-sprite-kit/`.

8.2 Color Maze

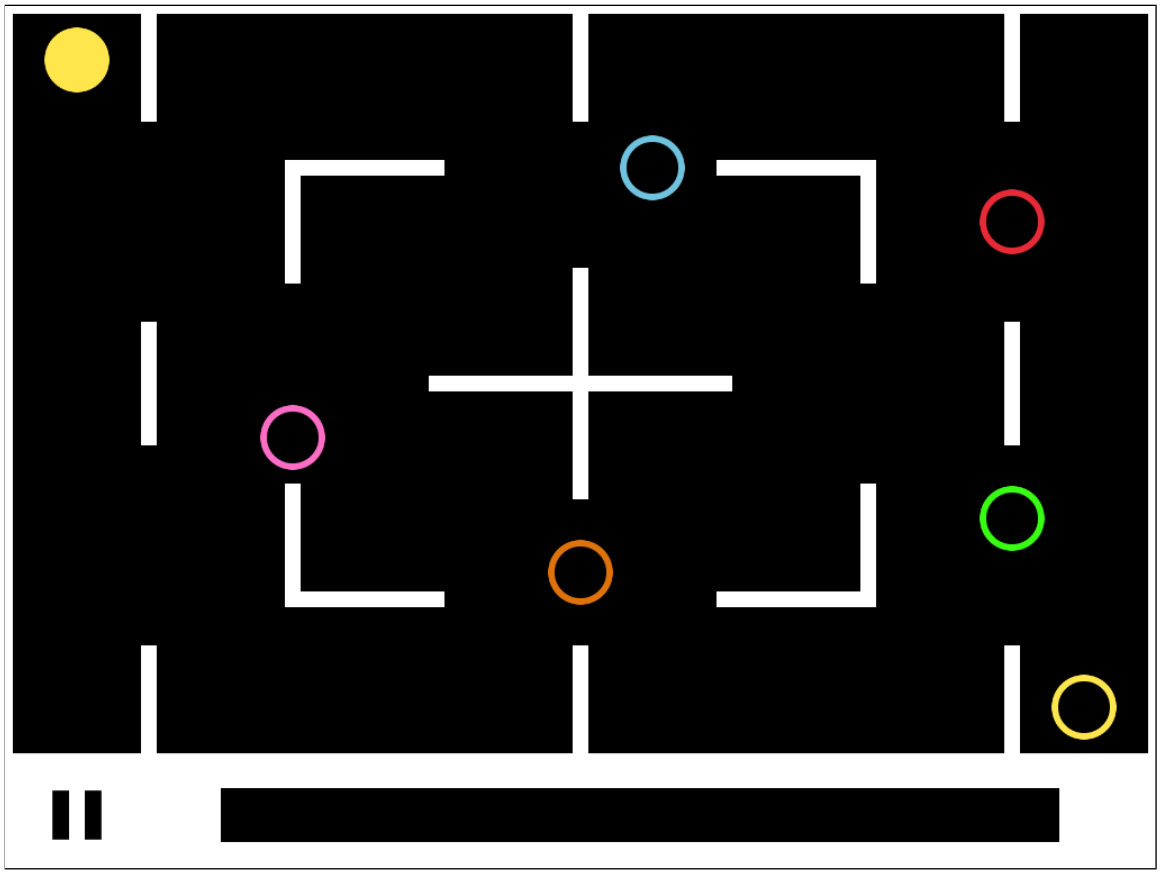
Moja druga iOS igra je "Color Maze". Tu igru sam radio u sklopu mog trećeg nastupa (2014.) na App Start Contestu. Cilj igre je na 60 jedinstvenih levela u određenom vremenu

[3] `https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html`



Slika 8.2: Prve skice igre ColorTap!

kuglicu određene boje dovesti do kružnice iste te boje. Igrač koristeći uređaj, tj. gibanjem uređaja kako bi se aktivirale promjene u akcelerometru, pomiče kuglicu kroz labirint. Ukoliko igrač uspije dovesti lopticu do cilja u zadanom vremenu i bez da dodirne kružnice drugih boja, tada je uspješno završio level i osvaja zaslužene bodove i zvjezdice. Primjer jednog levela može se vidjeti na slici 8.3.



Slika 8.3: Primjer jednog od levela u igri Color Maze

Game Center

Obje igre ColorTap! i Color Maze imaju integriran Game Center[4], Appleov oblik društvene mreže za bilježenje i praćenje rezultata u igrama. U igri Color Maze Game Center koristim nešto više, na način da na početku svakog levela i na kraju svakog levela igraču pokažem najbolji rezultat (igračeva prijatelja na GC ili općenito najboljeg igrača) i tako se potiče natjecateljski duh igrača koji igra. Mnoge popularne igre kao što su "Doodle Jump" i "Temple Run" koriste tu metodu koja se pokazala dosta uspješnom. Primjer prikaza najboljeg rezultata ostalih igrača na kraju odigranog levela može se vidjeti na slici 8.4.

[4] `http://developer.apple.com/game-center/`

Slika 8.4: Primjer prikaza najboljeg rezultata ostalih igrača u igri Color Maze

Chartboost

Igra Color Maze je besplatna za skidanje, a za monetizaciju igre koristim mrežu za oglašavanje "Chartboost"[5]. Chartboost je specijalizirana mreža za oglašavanje namijenjena samo igrama. Chartboost koristim kako bih prikazivao oglase ostalih developera, ali i za promociju ostalih mojih igara (zasad samo još jedne - ColorTap!). Svim developerima koji rade igre preporučio bih Chartboost kako bi monetizirali svoje besplatne igre, a ako se rade "obične" aplikacije onda se mogu koristiti ostale mreže kao što su Appleov iAd[6], Googleov AdMob[7] itd. Primjer prikaza Chartboost oglasa može se vidjeti na slici 8.5.

[5] `http://chartboost.com`
[6] `http://developer.apple.com/iad/`
[7] `http://www.google.com/ads/admob/`

Slika 8.5: Primjer prikaza Chartboost oglasa u igri Color Maze

8.3 Sprite Kit

Sprite Kit je Appleov 2D game framework za iOS i OS X. Sprite Kit sadrži sve potrebno za izradu kvalitetne igre s odličnim performansama, sve za animaciju spriteova, simuliranje fizike, odlične sustave čestica (*particle systems*). Ono što se meni najviše sviđa je jednostavnost Sprite Kit API-ja koji omogućuje realizaciju vrlo kompleksnih animacija i efekata, a opet uz malo koda i odlične performanse. Sprite Kit umjesto nas izvodi OpenGL kod, a mi se fokusiramo na dizajn same igre i poboljšavanje gameplaya.

Mogao bih napraviti poseban diplomski rad samo o Sprite Kitu, ali radije bih se koncentrirao na neke zanimljivosti (uz kratke opise i primjere koda) koje bi čitatelja motivirale da se upusti u Sprite Kit. Osobama koje više zanima Sprite Kit savjetovao bih da pogledaju "Sprite Kit Programming Guide" [3].

Scene

Animacije i prikaz izvodi objekt klase `SKView`. Sadržaj igre je organiziran u scene, koje su reprezentirane objektom klase `SKScene`. Scena sadrži sve spriteove i ostali sadržaj koji se treba prikazati. Svaka scena implementira logiku za svaki frame i procesiranje sadržaja. U svakom trenutku prikazuje se jedna scena i sve dok se prikazuje konkretna scena izvršavaju se animacije i ostala logika za tu scenu.

Slijedi primjer jednostavne scene i njenog prikazivanja.

```
// Konfiguracija u viewDidLoad metodi view controllera:
#import <SpriteKit/SpriteKit.h>

- (void)viewDidLoad
{
  [super viewDidLoad];
  SKView *spriteView = (SKView *) self.view;
  spriteView.showsDrawCount = YES;
  spriteView.showsNodeCount = YES;
  spriteView.showsFPS = YES;
}
```

Pretpostavimo da se koristi scena `HelloScene`, tada ju se može prezentirati na ovaj način:

```
#import "HelloScene.h"

- (void)viewWillAppear:(BOOL)animated
{
  HelloScene* hello = [[HelloScene alloc] initWithSize:CGSizeMake
      (768,1024)];
  SKView *spriteView = (SKView *) self.view;
  [spriteView presentScene: hello];
}
```

Sceni dodajemo još neke detalje za prikazivanje sadržaja. Kreiranje sadržaja radimo u metodi `didMoveToView` pozivom metode `createSceneContents` ukoliko sadržaj nije kreiran.

```
#import "HelloScene.h"

@interface HelloScene ()
@property BOOL contentCreated;
@end
```

```
@implementation HelloScene

@end

// implementacija didMoveToView: metode:
- (void)didMoveToView: (SKView *) view
{
  if (!self.contentCreated)
  {
    [self createSceneContents];
    self.contentCreated = YES;
  }
}
```

Metoda za kreiranje sadržaja:

```
- (void)createSceneContents
{
  self.backgroundColor = [SKColor blueColor];
  self.scaleMode = SKSceneScaleModeAspectFit;
  [self addChild: [self newHelloNode]];
}
```

Pomoćna metoda za kreiranje čvora:

```
- (SKLabelNode *)newHelloNode
{
  SKLabelNode *helloNode = [SKLabelNode
labelNodeWithFontNamed:@"Chalkduster"];
  helloNode.text = @"Hello, World!";
  helloNode.fontSize = 42;
  helloNode.position = CGPointMake(CGRectGetMidX(self.frame),
        CGRectGetMidY(self.frame));
  helloNode.name = @"helloNode";
  return helloNode;
}
```

Prelazak iz prikaza jedne scene u prikaz druge scene je vrlo jednostavan. Pretpostavljamo da postoji neka scena tipa `SpaceshipScene`, na sljedeći način možemo preći u tu scenu iz trenutne scene:

```
SKScene *spaceshipScene  = [[SpaceshipScene alloc] initWithSize:
      self.size];
SKTransition *doors = [SKTransition doorsOpenVerticalWithDuration
      :0.5];
```

```
[self.view presentScene:spaceshipScene transition:doors];
```

Node Tree

Glavna klasa (i njene podklase) koju se koristi u Sprite Kit igri je `SKNode` (čvor, *node*), čak je i `SKScene` njena podklasa. Scenu se gradi tako da se gradi stablo čvorova (*node tree*) na način da se čvorovima dodaju čvorovi kao djeca i tako se formira cijela scena. Postoje razni tipovi čvorova, neki se koriste za prikazivanje sadržaja, a neki samo za strukturu stabla čvorova kako bi se napravila hijerarhija čvorova scene.

Primjeri klasa čvorova:

- `SKNode` - čvor koji sam po sebi ne isrctava ništa, sve ostale klase čvorova nasljeđuju ovu klasu
- `SKScene` - klasa čvora koja procesira sve akcije i animacije, ujedno i korijen u stablu čvorova
- `SKSpriteNode` - čvor koji iscrtava sprite
- `SKLabelNode` - čvor koji iscrtava tekst
- `SKShapeNode` - čvor koji iscrtava određeni geometrijski oblik (koristeći Core Graphics path)
- `SKVideoNode` - čvor koji prikazuje video sadržaj
- `SKEmitterNode` - čvor koji stvara i iscrtava čestice
- `SKCropNode` - čvor koji reže određeni čvor ovisno o masci
- `SKEffectNode` - čvor koji primjenjuje određeni efekt na svu svoju djecu čvorove

Akcije

Sadržaj scene najlakše je animirati korištenjem akcija. Svaka akcija koja se koristi je objekt klase `SKAction`. Najčešće se akcije koriste kako bi se animirale promjene svojstava čvorova (npr. lokacija, boja, veličina itd.). Mogu se kombinirati razne akcije i kreirati kompleksne akcije kako bi se postigli željeni rezultati. Kreirane akcije jednostavno se pozivaju/pokreću na čvorovima.

Primjer s `HelloScene` proširujem na način da dodajemo akcije. Reagiranjem na dodir ekrana, u metodi `touchesBegan:withEvent:` kreiramo akciju koja pomiče, povećava i kasnije uklanja čvor.

```
- (void)touchesBegan:(NSSet *) touches withEvent:(UIEvent *)event
{
  SKNode *helloNode = [self childNodeWithName:@"helloNode"];
  if (helloNode != nil)
  {
    helloNode.name = nil;
    SKAction *moveUp = [SKAction moveByX: 0 y: 100.0 duration:
        0.5];
    SKAction *zoom = [SKAction scaleTo: 2.0 duration: 0.25];
    SKAction *pause = [SKAction waitForDuration: 0.5];
    SKAction *fadeAway = [SKAction fadeOutWithDuration: 0.25];
    SKAction *remove = [SKAction removeFromParent];
    SKAction *moveSequence = [SKAction sequence:@[moveUp, zoom,
        pause, fadeAway, remove]];
    [helloNode runAction: moveSequence];
    }
}
```

Akcije se mogu kreirati kombinacijom drugih akcija na način da ih se grupira (*groups*) ili stavlja u slijed (*sequence*). Ako se akcije grupiraju onda svaka akcija u grupi počinje s izvođenjem u isto vrijeme, a grupna akcija je duljine najdulje pojedinačne akcije. Ako se akcije stave u slijed onda se akcije izvode jedna za drugom i cijela akcija je duljine zbroja duljina trajanja svih pojedinačnih akcija. Svaka akcija napravljena grupiranjem ili slijedom može se opet grupirati ili stavljati u slijed i tako se mogu raditi kompleksne akcije ovisno o potrebama.

Fizika

Sprite Kit sadrži 2D *physics engine* koji omogućuje simuliranje fizike u scenama u igri. Često igre koje radimo imaju elemente koji imaju međusobnu interakciju, sudaraju se itd. Kreiranjem fizikalnog tijela (*physics body*), točnije korištenjem klase `SKPhysicsBody` čvoru možemo dodati fizikalna svojstva. Na taj način čvor će reagirati na simuliranje gravitacije (koje možemo također imati podešeno), na sudaranja s ostalim čvorovima itd.

Vrlo jednostavno se može nekom čvoru dodati fizikalna svojstva. Dovoljno je samo kreirati `SKPhysicsBody` za taj čvor:

```
node.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:node.
    size];
```

Vrlo jednostavno se može napraviti efekt padanja kamenja na način da se kreira akcija koja se izvodi neprestano i stvara kamen koji pod utjecajem simulacije gravitacije pada i

kad se kamen makne s ekrana kamen se ukloni iz scene:

```
static inline CGFloat skRandf()
{
  return rand() / (CGFloat) RAND_MAX;
}
static inline CGFloat skRand(CGFloat low, CGFloat high)
{
  return skRandf() * (high - low) + low;
}
//
- (void)addRock
{
  SKSpriteNode *rock = [[SKSpriteNode alloc] initWithColor:[
      SKColor brownColor] size:CGSizeMake(8,8)];
  rock.position = CGPointMake(skRand(0, self.size.width),self.
      size.height-50);
  rock.name = @"rock";
  rock.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:rock.
      size];
  rock.physicsBody.usesPreciseCollisionDetection = YES;
  [self addChild:rock];
}
// kamen uklanjamo kad se makne s ekrana:
-(void)didSimulatePhysics
{
  [self enumerateChildNodesWithName:@"rock" usingBlock:^(SKNode *
      node, BOOL *stop)
  {
    if (node.position.y < 0)
      [node removeFromParent];
  }];
}
```

Video

Na vrlo jednostavan način se može u igru dodati video sadržaj koristeći `SKVideoNode`. Na taj način video čvor se tretira kao svaki drugi čvor i na njega se mogu primjenjivati ostale animacije, akcije itd. Slijedi jednostavan primjer kreiranja video čvora:

```
SKVideoNode *sample = [SKVideoNode videoNodeWithVideoFileNamed:@"
    sample.m4v"];
```

```
sample.position = CGPointMake(CGRectGetMidX(self.frame),
[self addChild: sample];
[sample play];
```

Zašto Sprite Kit?

Postoje razni načini za razvoj igara. Neki developeri su više iskusni neki manje, neki vole pisati kod na nižoj razini (OpenGL), neki na višoj. Ja bih Sprite Kit preporučio svima koji žele raditi jednostavne, ali i nešto složenije 2D igre, ne žele se previše upuštati u to kako se sva iscrtavanja, simulacije fizike i slične složene stvari efektivno rade. Korištenjem Sprite Kita developer ima garanciju da će na uređajima s iOS i OS X operacijskim sustavima imati najbolje performanse i da se Apple brine da sve naprosto funkcionira. Ja imam odlična iskustva sa Sprite Kitom i preporučio bih ga svakom developeru koji nije još napravio niti jednu igru, jer je stvarno ugodno i zabavno raditi sa Sprite Kitom.

Kao što sam već rekao, Sprite Kit omogućuje zaista još mnoštvo raznih stvari, ali nema smisla ići previše u dubinu. Za detaljnije informacije tu je službena Apple dokumentacija i Sprite Kit vodič [3].

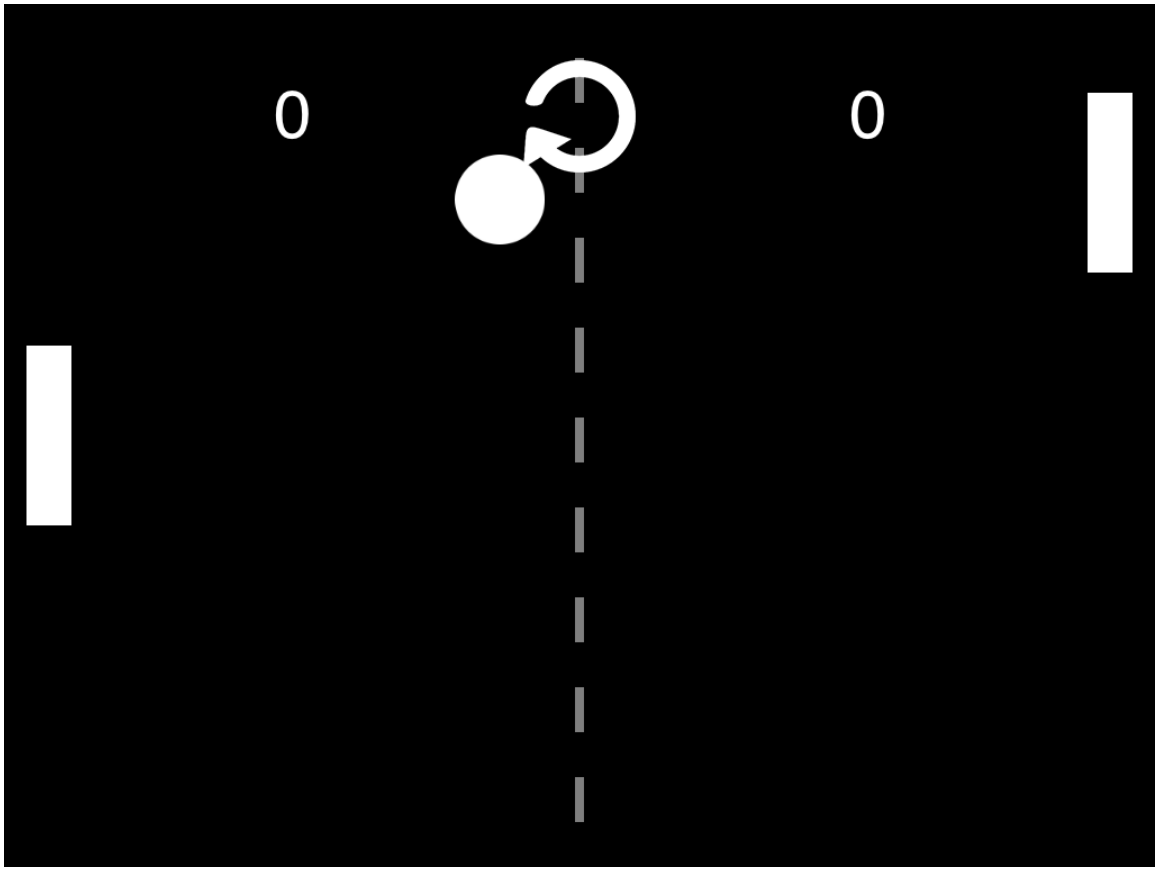
8.4 Open source

Mnogi developeri sudjeluju u open source projektima ili rade na nekim svojim projektima koje onda stave kao open source, primjerice na Github[8]. Ja nisam previše aktivan u open sourceu, ali trenutno na svom Github profilu imam objavljenu jednu Objective-C kategoriju koju sam napravio za jedan svoj projekt i napravio sam sa Sprite Kitom verzije dvije vrlo popularne igre (Pong i Breakout). Moj Github profil s open source projektima nalazi se na ovoj stranici `http://github.com/bozidarsevo/`.

Sprite Kit Pong

Pong[9] je jedna od prvih arkadnih video igara i jedna od sigurno najpopularnijih. Igru je izradila kompanija Atari[10]. S obzirom koliko je Sprite Kit jednostavan i moćan, verziju Ponga sam napravio u jednom danu. Cilj izrade te igre pomoću Sprite Kita bio je da pokažem koliko je Sprite Kit jednostavan i kako se uz malo koda mogu jako moćne i zanimljive stvari napraviti. A pošto je Pong vrlo popularna igra odlučio sam napraviti njezinu verziju. Slika 8.6 pokazuje primjer igranja ove verzije igre.

[8] `http://github.com`
[9] `http://en.wikipedia.org/wiki/Pong`
[10] `http://atari.com`



Slika 8.6: Primjer igranja Sprite Kit Pong verzije

Moju verziju Pong igre može se pogledati na Github repozitoriju: `https://github.com/bozidarsevo/sprite-kit-pong`.

Sprite Kit Breakout

Breakout[11] je također vrlo popularna arkadna igra koju je napravila kompanija Atari, točnije koju je napravio Steve Wozniak (uz Stevea Jobsa). O detaljima razvoja te igre može se pročitati i u Wozniakovoj knjizi "iWoz" [5] gdje Wozniak priča o raznim zanimljivostima iz svog života pa tako i o razvoju igre Breakout. Nakon što sam napravio Sprite Kit verziju igre Pong odlučio sam napraviti i verziju igre Breakout. Breakout je u biti primjer Pong

[11] `http://en.wikipedia.org/wiki/Breakout_(video_game)`

igre, ali za jednog igrača. Tako da je bilo logično uz Pong napraviti i verziju Breakout igre. Slika 8.7 pokazuje primjer igranja ove verzije igre.

Slika 8.7: Primjer igranja Sprite Kit Breakout verzije

Moju verziju Breakout igre može se pogledati na Github repozitoriju: `http://github.com/bozidarsevo/sprite-kit-breakout`.

Bibliografija

[1] Apple Inc., *iPhone OS Programming Guide*, https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html, 2014, [Online; pristupljeno 26.8.2014].

[2] ———, *Programming with Objective-C*, https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html, 2014, [Online; pristupljeno 26.8.2014].

[3] ———, *Sprite Kit Programming Guide*, https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html, 2014, [Online; pristupljeno 28.8.2014].

[4] W. Isaacson, *Steve jobs*, Simon and Schuster, 2011.

[5] S. Wozniak and G. Smith, *iwoz*, W. W. Norton and Company, 2006.

Sažetak

U ovom diplomskom radu iznosim osobna iskustva vezana za razvoj iOS aplikacija i osnove razvoja za iOS platformu. U prva dva poglavlja iznosim neke povijesne i trenutne činjenice o Apple Inc. (kompaniji koja stoji iza operacijskog sustava iOS i uređaja koji koriste taj operacijski sustav), iPhoneu i iOS-u.

Nakon početnog dijela koncentriram se na Objective-C, glavni programski jezik koji se koristi pri razvoju iOS aplikacija. U tom poglavlju se bavim glavnim značajkama Objective-C jezika, npr. kako definirati klasu, korištenje blokova, koje su konvencije prisutne itd.

Razvoj iOS aplikacija je zanimljiv proces tako da u četvrtom poglavlju iznosim metode koje koristim pri razvoju iOS aplikacija i svoj pristup cijelom razvojnom procesu. Da bi developer testirao aplikacije na osobnom uređaju i objavio aplikacije u App Storeu potrebno je platiti godišnju developer licencu. Tom temom i ostalim informacijama vezanim za objavu aplikacija u App Storeu se bavim u petom poglavlju. U šestom poglavlju se bavim najboljim kolegijem za učenje razvoja iOS aplikacija "CS139p" koji se održava na sveučilištu Stanford. Snimke tih predavanja su dostupne svima za skidanje i najbolji su početak za svaku osobu zainteresiranu za razvoj iOS aplikacija.

U zadnja dva poglavlja bavim se aplikacijama i igrama koje sam u zadnjih 4 godine razvio i objavio u App Storeu. U poglavlju o igrama bavim se i Sprite Kitom, frameworkom za razvoj 2D igara za iOS i Mac OS X.

Summary

In this thesis I share personal experiences related to the development of iOS applications and fundamentals of development for iOS platform. In the first two chapters I bring up some historical and current facts about Apple Inc. (the company behind the iOS operating system and devices that use the operating system), about iPhone and iOS.

After the initial portion I concentrate on Objective-C, the main programming language used when developing iOS applications. In this chapter, I explain the main features of the Objective-C language, for example how to define a class, use blocks, which conventions are being used, etc..

Developing iOS application is an interesting process so in the fourth section I present the methods I use when developing iOS applications and my approach to the entire development process. To test the application on his device and publish applications on the App Store, developer is required to pay an annual developer license. I cover that topic and other information related to the publication of applications on the App Store in fifth chapter. In the sixth chapter, I talk about the best course for learning iOS application development "CS139p" which is held at Stanford University. Recordings of these lectures are available to everyone for downloading and they are the best start for any person interested in developing iOS applications.

In the last two chapters, I deal with applications and games that I have made and released on the App Store in the last four years. In the chapter about the games I also talk about Sprite Kit, framework for 2D game development for iOS and Mac OS X.

Životopis

Rođen 19.3.1990. u Zagrebu, živim u Samoboru. Razvijam iOS aplikacije sad već otprilike 4 godine, u kojima sam objavio 13 aplikacija na App Storeu, od kojih su 2 igre. Zanima me razvoj web i mobilnih aplikacija. U zadnje vrijeme sam se dosta zainteresirao za Sprite Kit i razvoj igara sa Sprite Kitom. Također krenuo sam i u razvoj web aplikacija uz pomoć Ruby on Rails frameworka.

Zanimaju me razne stvari, tako sam bio državni prvak u latinoameričkim i standardnim plesovima, volim snimati i producirati video materijale, razvijati software, baviti se sportom itd. San/cilj mi je jednog dana imati vlastitu tvrtku koja će se baviti razvojem softvera, igara, video produkcijom itd.

WEB PRISUTNOST

- moj portfolio i glavni website - `http://www.seodoa.com`
- open source - `http://github.com/bozidarsevo`
- Twitter `http://twitter.com/bsevo`
- Instagram - `http://instagram.com/bsevo`
- 500px - `http://500px.com/bsevo`
- Linkedin - `http://linkedin.com/in/bozidarsevo`
- video produkcija - `http://www.youtube.com/user/bsevo19`
- video produkcija - `http://www.youtube.com/user/seodoadev`
- video produkcija - `http://www.youtube.com/user/croatiapple`
- video produkcija - `http://vimeo.com/seodoa/`

ISKUSTVO

2014

- App Start Contest - finalist s iOS igrom “Color Maze” - napravljena sa Sprite Kitom

2013

- volonter na Webcamp konferenciji, photo, video i ostale standardne stvari u pripremi lokacije (hrana, piće) - http://2013.webcampzg.org
- Apps World London 2013, Indie Game Zone - izlagač s igrom ColorTap!
- produkcijski partner na Spark.me konferenciji u Budvi - `http://backway.me/sparkme`
- ColorTap - iOS igra napravljena sa Sprite Kitom - `http://colortap.me`
- Fresh Island Festival aplikacija - `http://backway.me/freshisland`
- mentor na iOS development radionici
- App Start Contest - Osvojio 1. nagradu (10000 HRK) s “ZABA4U” iOS aplikacijom za partnersku temu “Mobilno bankarstvo” Zagrebačke banke, `http://zaba.hr`
- Shift Conference - Challenge finalist s “BackWay” - `http://shiftsplit.com`
- Business Plan Contest finalist (u top 11) - `http://www.estudent.hr/bpc`
- posjetio OMGcommerce conference - `http://omgcommerce.me`
- prezentirao BackWay na “SMEs of the future - by United Kingdom Trade & Investment Croatia”
- Startup Live Zagreb(`http://startuplive.in/zagreb/1/`) - s BackWay projektom
- sudjelovao na Erste 24h Finnovation Hackathonu (tim Seodoa) i osvojio 4. mjesto s aplikacijom/igrom “ErsteGrad”.
- član ZIP-a (zagrebački inkubator poduzetništva) s projektom BackWay

2012

- sudjelovao u Best Code Challengeu - mobilna aplikacija + web/server
- jedini hrvatski predstavnik i predstavljao BackWay na "European Space Solutions conference" u Londonu - `http://www.space-solutions.eu`
- prezentirao BackWay na "Geeks on a Plane" eventu u Zagrebu
- Webfest Webup finalist, Budva, Montenegro
- osvojio 2. nagradu na "Startup Camp Vis" s projektom BackWay, 27.-31.8.2012.
- aplikacija BackWay nominirana kao hrvatski kandidat za "UN World Summit Award Mobile 2012"
- App Start Contest u top 20 aplikacijama s iPhone aplikacijom BackWay

2011

- Case Study Competition Finalist (NovaTV)
- dvije epizodne uloge u TV Sudnici - `http://bit.ly/Y8zCns`

VJEŠTINE

- Mac OS, iOS, Linux, Windows
- Objective C, Sprite Kit, C, C++, C#, HTML, PHP, Javascript, MySQL, SQLite, JSON, Git, Ruby on Rails
- Apple iWork, Apple iLife, Xcode, Final Cut Pro, Aperture, After Effects, Photoshop
- hrvatski, engleski, njemački

OBRAZOVANJE

Diplomski studij - računarstvo i matematika

PMF-MO, Sveučilište u Zagrebu, 2011. - danas

- dvije akademske godine (tj. četiri semestra), 120 ECTS bodova

- prva godina: Građa računala, Oblikovanje i analiza algoritama, Umjetna inteligencija, Računalna grafika, Uvod u paralelno računanje, Društveni aspekti ICT-a, Operacijski sustavi, Izračunljivost, Baze podataka, Računarski praktikum 2, Konačne geometrije, Matematički softver, Izrada web projekta (FER vještina - App Start Contest)

- druga godina: Interpretacija programa, Softversko inženjerstvo, Kriptografija i sigurnost mreža, Računarski praktikum 3, Meta-heuristike, Multimedijski sustavi, Složenost algoritama, Distribuirani procesi, Upravljanje softverskim projektima, Strojno učenje

- http://www.math.pmf.unizg.hr/Default.aspx?art=4104&sec=465

Preddiplomski studij - matematika

PMF-MO, Sveučilište u Zagrebu, 2008. - 2011.

- tri akademske godine (tj. šest semestara), 180 ECTS bodova

- sveučilišni prvostupnik (baccalaureus) matematike, univ. bacc. math.

- prva godina: Matematička analiza 1 & 2, Linearna algebra 1 & 2, Elementarna matematika 1 & 2, Programiranje 1 & 2, Tjelesna i zdravstvena kultura 1 & 2

- druga godina: Diferencijalni račun funkcija više varijabli, Diskretna matematika, Vjerojatnost, Strukture podataka i algoritmi, Engleski jezik struke 1 & 2, Tjelesna i zdravstvena kultura 3 & 4, Matematičko modeliranje u biologiji, Integrali funkcija više varijabli, Algebarske strukture, Numerička matematika, Računarski praktikum 1, Bioinformatika

- treća godina: Mreže računala, Obične diferencijalne jednadžbe, Vektorski prostori, Statistika, Matematička logika, Kompleksna analiza, Teorija skupova, Metode matematičke fizike, Mjera i integral, Objektno programiranje (C++)

- http://www.math.pmf.unizg.hr/Default.aspx?art=4109&sec=466

Srednja škola

Gimnazija A. G. Matoša, Samobor, 2004. - 2008.

Osnovna škola

OŠ Bogumil Toni, Samobor