**Tallinna Ülikool**

**Informaatika Instituut**

# Õppematerjali koostamine töölauarakenduste loomiseks Mac OS X platvormile

**Bakalaureusetöö**

Autor: Lauri Roomere

Juhendaja: Andrus Rinde

Autor: .....,,.....,,2014

Juhendaja: .....,,.....,,2014

Instituudi direktor: .....,,.....,,2014

Tallinn 2014

# Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

| | |
|---|---|
| ..... | ..... |
| (kuupäev) | (autor) |

# Sisukord

# Sissejuhatus

Mac OS X platvormile rakendusi loovatest tarkvara arendajatest, programmeerijatest on autorile teadaolevalt Eestis ja Euroopas puudus. Autorile on teada juhtumeid, kus pole mitme aasta jooksul õnnestunud tööle sobivat arendajat palgata.

Eesti kõrgkoolides koolitatakse arendajaid pigem PC-platvormi silmaspidades, Mac OS X tarkvara arenduse põhimõtteid praktiliselt ei käsitleta. Leidub õppematerjal rakenduste loomisest iOS operatsioonisüsteemi kasutavatele seadmetele, kuid Mac OS X'i operatsiooni süsteemi kasutavatele arvutitele eestikeelset õppematerjali autori teada loodud ei ole.

Kõige suurem erinevus Mac OS X'ile ja Windows'ile rakendusi luues tuleneb operatsiooni süsteemide erinevusest ja vastavatele operatsiooni süsteemidele mõeldud arendusvahendites, just omaste (ingl.k *native)* rakenduste valmistamist silmas pidades.

Autor töötab arendajana Mac platvormile ja on pidanud kõik oskused omandama iseseisvalt, materjale lugedes ning läbi töötades. Eestikeelseid abimaterjale pole üldse leidnud, mis viitab vajadusele luua õppematerjali Mac platvormile tarkvaraarenduse iseärasusi silmaspidades.

Käesoleva bakalaureusetöö peaeesmärgiks on koostada õppematerjal, mille abil oleks võimalik omandada esmased oskused Mac OS X platvormile rakenduste valmistamiseks, kasutades selleks järgnevaid vahendeid:

- Xcode 5.0 tarkvaraarenduskeskkonda
- Cocoa ja Foundation raamistikku
- FMDB teeki
- SQLite Manger'i
- XCTest raamistikku
- Objective-C keel

Lisaks soovib autor tuua oma kogemuste põhjal näiteid ja soovitusi selle kohta, kuidas töölauarakendusi sellele platvormile valmistada ning milliseid vahendeid selleks kasutada. Samuti soovib autor tekitada huvi Mac OS X platvormile rakenduste valmistamise vastu.

Eesmärgi saavutamiseks tutvub autor esmalt olemasolevate õppematerjalide ja kursustega, mis on suunatud Mac OS X platvormil töölaua rakenduste arendamisele. Õppematerjalis käsitletavad inglisekeelsetes materjalides ja Tallinna Ülikooli .Net raamistiku ainekursuses käsitletavad teemad ja lood on võetud eeskujuks materjali struktuuri koostamisel. .Net raamistiku abil on

võimalik tuua välja ka arendamise protsessis olevaid erinevusi Cocoa ja .Net raamistike vahel. Autor järgib ka ADDIE mudelit õppematerjali loomisel nii palju kui võimalik.

Esimeses peatükis tutvustatakse Mac OS X'ile arendamiseks vajaminevaid vahendeid, antakse lühike ülevaade olemasolevatest inglisekeelsetest materjalidest, mis on autori enda poolt läbi töötatud. Autor annab ülevaate hetkeseisust vastavale platvormile arendamisele suunatud õppimise võimalustest Eestis.

Teises peatükis kirjeldab autor täpsemalt õppematerjali loomiseks kasutatavaid arendus vahendeid ja toob välja teisi võimalusi kuidas Mac OS X platvormile rakendusi luua. Samuti üritab autor välja tuua Mac OS X'il Cocoa raamistiku ja Windows'il .Net raamistiku erinevused rakenduste valmistamisel.

Kolmandas peatükis põhjendab autor oma materjali üleseehitust ning määrab sihtrühma, kellele on õppematerjal loodud ja millised eelnevad teadmised peaksid olema, et õppematerjali iseseisvalt läbida. Samuti määratakse ka oodatavad õpitulemused, mida materjali läbinu saavutama peaks.

# Bakalaureusetöös kasutatud mõisted

*XCode* arenduskeskkond - Apple poolt loodud arenduskeskkond, rakenduste arendamiseks Mac OS X ja iOS platvormile.

*Microsoft Visual Studio* - Microsofti poolt loodud arenduskeskkond, mille abil on võimalik luua rakendusi Windows'i platvormile.

*NuGet* – NuGet on Microsoft'i arendus platvormile loodud pakettide haldur. NuGet võimaldab luua ja tarbida NuGet'i pakkete (Outercurve Foundation , 2014).

*CocoaPods*- *CocoaPods* on Objective-c projektide haldus vahend. See sisaldab tuhandeid teeke ja aitab projekti mahtu ning hallatavaid teeke kontrolli all hoida (CocoaPods, 2014).

*Mono* – Avatud lähtekoodiga ja vabavaraline .Net raamitikuga ühilduv arendusvahend mitmeplatvormiliste rakenduste loomiseks Windows platvormil, lähemalt saab lugeda Mono kohta järgneval aadressil: http://www.mono-project.com.

*MVC* – tarkvara arhitektuur, mis jagab andmed mudeli, vaate ja kontrolleri vahel.

# 1 Ülevaade programmeerimiskursustest, mis pakuvad ettevalmistust Mac platvormile tarkvara arenduseks

Käesolevas peatükis analüüsib autor erinevates Eesti kõrgkoolides pakutavaid programmeerimis-kursuseid ning annab lühikese hinnangu inglisekeelsetest õppematerjalidest, millega on tutvunud ja kuidas need aitavad ette valmistada arendajaid Mac platvormile. IT Kolledzi õppekavas, pakkus ainsana ainet, mis on seotud Mac platvormile arendamisega, kuid mitte Mac'ile töölauarakenduste arendamisele. Bakalaureuse õppekavad valis autor uurimiseks seetõttu, et ta ise õpib Tallinna Ülikoolis Informaatika Instituudis tarkvara arenduse suunal ning oli huvitatud, kas mõnes teises kõrgkoolis on ainet, mis käsitleks vastavat teemat ning millisele suunitlusele on pigem nende ainekavad keskendunud.

## 1.1 Ülevaade inglisekeelsetest materjalidest

Internetist on kättesaadavad ka mitmed raamatud, mille abil ennast Mac'ile rakenduste arendamisega kurssi viia. Siinkohal toob autor välja mõned materjalid, millega ta on lähemalt tutvunud või tööprotsessis kokku puutunud ning mis õppurile abiks oleksid.

**„Objective-C Programming: The Big Nerd Ranch Guide (2nd Edition)"**

Autorid:Aaron Hillegass ja Mike Ward

Väljaandmise aasta: 2013

See raamat oli üks esimesi, mille abil autor õppis arendamist Mac OS X platvormile. Raamat annab algajale esmased vajalikud teadmised programmeerimisest Mac OS X platvormile ning puudutab natuke ka iOS'ile arendamist.

Raamat käsitleb järgnevaid teemasid:

- Programmeerimise alused:muutujad,tsüklid, funktsioonid jne.
- Objektid, klassid, meetodid ja väljundid.
- Viidad, aadressid ja mäluhaldus
- Xcode'i, Apple dokumentatsiooni ja teiste arenduseks vajalike tööriistade kasutamine
- Foundation raamistiku klassid
- ARC ja retain tsüklid.
- Atribuudid

- Blokid
- Kategooriad
- Delegaadid, sündmused ja teavitus mustri disain
- Kasutatav Xcode 4.2, iOS 5 ja Mac OS X 10.7-ga

Autor soovitab algajatele, kes soovivad alustada programmeerimist ning neile, kes soovivad lihtsalt Objective-c keeles rakendusi looma hakata. Soovitatav on hankida kõige uuem versioon sellest raamatust, et oleks lihtsam uuema Xcode'i versiooniga raamatu ülesandeid lahendada ja juhendeid järgida. Raamatut saab osta Big Nerd Ranch'i kodulehelt aadressil :https://www.bignerdranch.com/book/objectivec_programming_the_big_nerd_ranch_guide

**„Cocoa and Objective-C Cookbook"**

Autor:Jeff Hawkins

Väljaandmise aasta: 2011

Autor kasutas ise seda raamatut oma teadmiste täiendamiseks Mac OS X platvormile rakenduste arendamisel. Teoreetilisem raamat kui „Objective-C Programming: The Big Nerd Ranch Guide (2nd Edition)". Räägib rohkem sellest, kuidas rakendusi luua ning lisaks tutvustab ka MySQL ja SQLite kasutamist oma rakenduses. Soovitatav juba edasijõudnutele. Raamat on ostetav aadressilt: http://www.packtpub.com/cocoa-and-objective-c-cookbook/book

**AppleProgramming**

YouTube'i kanal, mis sisaldab videojuhendeid rakenduste loomiseks Cocoa ja Xcode'iga. Soovitan inimestele, kes omandavad teadmisi paremini videojuhendeid järgides. Võrreldes raamatutega on need videod pealiskaudsemad ja sisaldavad vähem detailset infot.

Aadress:https://www.youtube.com/channel/UCDg-YmnNehm3KB0BpytkUJg

**Mac Developer Library**

Veebiraamatukogu sisaldab vajalikku informatsiooni erinevatest arendusvahenditest, teekidest, klassidest kuni hea disainimise tavadeni Mac OS X platvormile välja. Ühesõnaga sisaldab kõike vajalikku, et teha algust arendamisega Mac OS X platvormile ning ka edaspidiseks täienduseks. Kuid meeles tuleks pidada, et tegu ei ole õppematerjaliga, vaid suuremalt jaolt sisaldab see

dokumentatsiooni erinevate tehnoloogiate ja võimaluste kohta. Siiski leidub seal erinevaid kasulikke juhendeid ka Mac OS X'ile rakenduste arendamiseks (Apple Inc, 2013).

Soovitan kasutada juhul, kui on mõne kindla kasutajaliidese objekti, meetodi või üleüldse Mac OS X arendust puudutavate teemade kohta detailsemat informatsiooni saada.

Aadress:Mac Developer Library link

## 1.2 Tarkvaraarenduse käsitlemine erinevates õppekavades

Autor töötab firmas, mis arendab välja tarkvara Mac OS X platvormile ja talle teadaolevalt ei koolitata Eestis vastavale platvormile arendajaid/programmeerijaid, küll aga leidub kursuseid, mis õpetavad arendama iOS, iPad ja iTouch platvormile. Firma otsingute tulemusena vastavale positsioonile inimese leidmisest sai autor teada, et Euroopas üldiselt on väga raske leida Mac OS X platvormile arendajat.

Uurides erinevate ülikoolid õppekavasid, jõudis autor järeldusele, et autori poolt analüüsitud koolide õppekavad on pigem suunatud erinevat tüüpi veebirakenduste loomisele ning töölauarakenduste loomisele on vähem rõhku pandud.

Teiseks keskenduvad kursused peamiselt Microsoft Windows'i operatsiooni süsteemidele jättes Mac OS X platvormi käsitlemata. Enamjaolt kasutatakse töölaua rakenduste loomise õpetamiseks Java keelt, Eclipse arenduskeskkonda ja C#, Microsoft Visual Studio't ning .Net raamistikku. Ühestki ülikoolist ei leidnud autor spetsiifilist kursust, mis käsitleks rakenduse arendamist Mac OS X platvormile omases (ingl.k *native*) keeles.

Ainsana Apple platvormile keskenduva kursusena võib leida IT Kolledzi kursuse "Sissejuhatus Apple mobiilitehnoloogiatesse", aga see keskendub mobiilsete rakenduste loomisele mitte töölauarakenduste loomisele. Kursusel tutvustatakse ka Objective-c keelt ning Cocoa raamistikku, kursus on keskendunud Apple mobiilsetele seadmetele rakenduste arendamisega nagu iPhone, iTouch ja iPad. Ühtegi eestikeelset õppematerjali mis keskenduks töölaua rakenduste loomisele Mac OS X platvormile autor ei leidnud.

# 2 Mac OS X platvormi arendusvahendid ja nende erinevus Windows platvormi arendusvahenditega.

Selles peatükis toob autor välja milliseid vahendeid kasutades saab veel Mac OS X'ile rakendusi luua. Autor üritab välja tuua ka erinevused, mille poolest õppematerjalis käsitletavaid vahendeid oleks parem Mac OS X'i platvormile rakendusi luues kasutada, kui Mono raamistikku või mitmeplatvormilisi keeli ja nendega seotud arendus vahendeid kasutades.

## 2.1 Mono raamistiku ja mitmeplatvormiliste keelte kasutamine Mac OS X rakenduste valmistamiseks

Esimese võimalusena tooksin välja Mono raamistiku, mille abil on võimalik arendada Windowsi platvormil mitmeplatvormilisi rakendusi C# programmeerimiskeelt kasutades. Mono'ga arendatud rakendused, aga ei ole nii töökindlad ning erinevate tehnoloogiate kasutamise võimalus oma rakenduse arendamise jaoks on väiksem, kui seda omases (ingl.k native) keeles tehes.

See tähendab seda, et sellist viisi rakenduse loomiseks kasutades ei saa te kõiki soovituid teeke/raamistikke kasutada, sest nad ei pruugi olla ümber konverteeritud Mono raamistikuga kasutamise jaoks.

Mono raamistiku kasutamine on keeruline ja seetõttu oleks lihtsam luua rakendus omases keeles ning kasutajatele ning operatsiooni süsteemi toetavale kasutajaliidesele üles ehitada.

Võimalus on kasutada ka mitmeplatvormilisi programmeerimis keeli nagu näiteks Java või C++, kuid kasutajaliides ning teatud funktsionaalsuste, tehnoloogiate kasutamine ja võimalused on ikkagi väiksemad kui seda teha vastavale platvormile mõeldud vahendite ning tehnoloogiatega.

Objective-c keelt, Xcode'i arenduskeskkonda ning Cocoa raamistikku kasutades luuakse Mac'ile mõeldud kasutajaliidesega rakendusi, mida vastava platvormi tarbijad on harjunud kasutama ning, mis ühtivad vastava kasutaja keskkonna ning tegevustega. Samas on see ka kõige lihtsam viis arendada tarkvara Mac OS X platvormile, sest vastavad tööriistad on mõeldud töötama just selle platvormiga ja loodud spetsiaalselt vastavale platvormile tema iseärasusi arvestades.

## 2.2 Erinevused Windows ja Mac OS X platvormile arendamises

Windowsi ja Mac OS X'ile arendamise suurimateks erinevusteks on just operatsioonisüsteemide omased programmeerimiskeeled ja raamistikud. Kuid on teatud iseärasused, mis muudavad just Mac OS X'il arendamise mugavamaks ja kiiremaks kui Windows'i platvormile.

Xcode'i saab kergesti ja kiiresti installeerida, installeerimisel kaasatakse koheselt vajalikud SDK'd ja teegid arendamiseks ja kui mõni vanem versioon on puudu, saab selle kergesti alla laadida läbi terminaali, samas kui Windows'il võib mõne SDK lisamisel kuluda tunde peamiselt näiteks operatsioonisüsteemide erinevustest tekkinud konfliktide lahendamisel. Selliste konfliktidega ei ole autor Mac OS X platvormil olevate vahenditega kokku puutunud.

Keerulisemaks teeb ka Windows'il arendamise erinevatest operatsioonisüsteemidest tulenevad probleemid ja nõuded, näiteks kui tahad Windows Phone 8 peale mobiili rakendust luua, pead ka vastavat operatsioonisüsteemi oma arvutil kasutama, millega sa rakendust arendad. Seda probleemi Xcode'i kasutades ei ole, vaja on ainult allalaadida õige SDK ja määrata projekti seadetes, millisele operatsioonisüsteemile ja seadele tarkvara luuakse.

Samamoodi nagu Windows'i arendajad kasutavad erinevate teekide ja raamistike haldamiseks NuGet'i, kasutavad Mac'i arendajad CocoaPod'i. Kui Microsoft Visual Studio's kasutatakse Windowsi Team Foundation Server'it versiooni kontrolliks, siis Xcode'is kasutatakse selleks sisse integreeritud Git'i . Soovi korral on võimalik ka teisi versioonihaldus vahendeid nagu näiteks SVN'I kasutada. Lisaks on autor kuulnud, et Windows on arendamas Git-TFS'i, mis on TFS kuhu on integreeritud ka Git sisse, mis muudaks Windows'i arendajal Mac'ile arendamisele üleminemise teatud määral lihtsamaks, sest TFS keskkonaga on arendaja juba tuttav.

Xcode'i arenduskeskkonda on integreeritud ka Profiler Instruments, mille abil saab testida rakenduse jõudlust, erinevate profileerimisvahendite abil ja avastada jõudluse ning mäluga seotud vead. Microsoft'i arendatud Microsoft Visual Studio's sellist vahendit aga ei ole ja vastavate testide läbiviimiseks tuleb kasutada mõnda välist vahendit.

Suurimad erinevused ongi kasutatavates operatsioonisüsteemides ja arendusvahendites, mõlemad kasutavad MVC (ingl.k Model-View-Controller) mustrit arendamisel ja omavad vajalikke vahendeid vastavatele platvormidele arendamiseks, kuid teatud kasutajaliideste ja arendamis-protsessi kuuluvate vahendite, tegevuste ja elementide nimetus on lihtsalt teistsugune (nt PopUpItem ja ComboBox). Objective-c süntaks võib olla algul harjumatu, kui inimene on pikka

aega arendanud Windows platvormil Java, C++, C# või mõnes muus sarnases keeles. Kuid kõik vajalik on olemas, et Objective-c keele ja raamistike abil lahendada probleeme ja ülesandeid, mida lahendatakse näiteks C#'i ja sellega seotud raamistikke kasutades.

Paljudele algajatele valmistab keerukust ja ebakindlust just Objective-C keele süntaks, mis on objektidele ülesehitatud. Kui C#'is deklareeritakse täisarvu lisades muutujale int tüübiga, siis Objective-C's on võimalik kasutada nii C keelest tulnud int tüüpi muutujat, kui Objective-C keele NSInteger objekti tüüpi muutujat, mis sisaldab ka vajalikke meetodeid täisarvuga töötamiseks. Objective-C keelt on võimalik koos C keelega kasutada.

Objective-C keelt ja Cocoa raamistikku kasutades aitab objektide mälus oleku haldamist kontrollida ARC, mis tegeleb automaatselt objektide lisamise ja vabastamisega mälust. Microsoft Visual Studio .Net raamistikus tegeleb vastavate toimingutga Garbage Collection.

Ebamugavaks muudab arendamise Mac OS X platvormile töölaua rakenduste loomiseks mõeldud juhendite ja õppematerjalide väike kogus, mis seletaks, kuidas keerukamaid või õppematerjalides mitte käsitletavaid probleeme lahendada. Rohkete juhendite hulk on Windows'i arendajatele eeliseks.

# 3 Õppematerjali loomine

Eesmärgiks on luua õppematerjal, mille abil oleks võimalik õppuril omandada vajalikud esmased teadmised Mac OS X platvormile töölaua rakenduste arendamisest kasutades selleks Cocoa raamistikku ja Xcode arenduskeskkonda.

Eesmärgi täitmiseks kasutab autor eeskujuna .Net raamistiku kursust TLÜ'st, keskendudes Cocoa raamistikuga Mac OS X platvormile rakenduste arendamisest ning arendamise võimalustest lähtudes ja jättes välja mobiilsete seadmetele rakenduste valmistamise Cocoa Touch raamistiku abil, kuna see väljub planeeritava õppematerjali teemade piiridest.
Vastava kursuse valis autor eeskujuks, sest vastav kursus tutvustab, kuidas arendada töölauarakendusi Microsofti platvormile nii-öelda native keeles (.Net raamistikku ning Visual Studio arenduskeskkonda kasutades) ning on idee poolest sobilik autori poolt soovitava õppematerjali loomisega. Autor valis nimetatud kursuse eeskujul teemad, mis käsitlevad süntaksit, andmebaasi liideseid ja sündmustele reageerimist. Teemade valikut mõjutasid ka inglisekeelsetes materjalides esmalt käsitletavad teemad. Autor soovib oma õppematerjalis tutvustada järgnevaid teemasid:

- andmete salvestamist SQLite andmebaasi ja FMDB teegiga
- lühidalt enim kasutatavaid kasutajaliidese objekte
- lühidalt Objective-c süntaksit
- Sündmustele/tegevustele reageerimist (delegaadid)

Omalt poolt lisab autor praktilisele kogemusele toetudes komponenttestimise peatüki, Mac OS X raamistikke (Cocoa ja Foundation raamistikke), Xcode'i ja ARC'i käsitlevad teemad. Need on teadmised, mida peab autor vajalikuks, et oleks võimalik programme luua Mac OS X platvormile ja rakenduse loomisprotsessi sujuvamaks muuta. Lisaks puutub autor igapäevaselt kokku antud teemades käsitlevate toimingutega. Nende vahendite abil on võimalik luua lihtsamaid rakendusi, mis suudavad andmeid salvestada, kuvada ja kasutajaliidese abil teatud valikuid teha.

## 3.1 Õppematerjali ülesehitus

Õppematerjali ülesehitusel tugineb autor õppurite vajadustel ning teadmistel, mida ootab temalt tööandja vastavale platvormile arendamist silmaspidades. Autor ehitab materjali üles vaadeldud kursuste, inglisekeelsete materjalide ja ADDIE mudeli eeskujul. Samuti püüab õppematerjali luues autor silmas pidada kaastöötajate poolt märgitud nõutavaid oskusi.

Vajalike oskuste välja selgitamiseks pöördus autor oma kaastöötajate ja ülemuse poole, et mõista, mida nemad ootavad uuelt arendajalt. Järgnevalt loetlen oskused, mida pidasid Mac OS X arendamise seisukohalt oluliseks:

- Objective-C keele oskus
- Oskus kasutada Xcode'i arenduskeskkonda
- SQL keele oskus
- Teadmised erinevatest tarkvaraarendamisega seotud disaini mustritest (nt. Singleton Pattern, Factory Pattern jne)
- Oskus kirjutada komponent teste
- Cocoa raamistikku ja teiste Mac OS X arendamisega seotud raamistike kasutamise oskus
- Põhilised teadmised programmeerimisest ehk programmerimise alused peaksid selged olema.

Autor alustab arendusvahendite tutvustusest ja seejärel asub koostama juhendeid, mille abil lõpuks õppurid suudaksid valmistada rakenduse, mis kasutab andmebaasi, tegevuste/sündmuste kuulamist ja erinevaid kasutajaliidese komponente.

Viimases peatükis olevad ülesanded on just keskendunud sellele, kui hästi õppur omandas õppematerjali ja vastavad teadmised. Ülesanded nõuvad kõikide õppetükkide läbimist nende lahendamiseks.

Autor valis materjali levitamise tüübiks pdf faili, sest sellele materjalile tuginedes oleks võimalik hiljem koostada vajalikke õppevideod ning animatsioone. Tekstifail on kergesti levitatav ning kõigile kättesaadaval.

Internetiühenduse puudumise korral on võimalik välja printida või alla laadida kohast, kus internet on kättesaadav ning printimine, allalaadimine lubatud/võimalik. Lisaks on õppematerjali lisatud temaatilisi joonistusi ning pilte, mille abil muutub õppematerjali sisu huvitamaks ja paremini arusaadavamaks. Teksti edasine muutmine ning ajakohastamine on samuti teistest meediumite tüüpidest kiirem ja lihtsam (Villems, et al., 2012).

Materjal on ühe tükina ja kehtib ühe (arenduskeskkonna,programmeerimiskeele) versiooni kohta ja on seega hiljem vähem segadust tekitav kui väljastatakse mõnest tehnoloogiast uus versioon.

Teemad on autor järjestanud nii, et alustatakse kergematest ülesannetest,teemadest ja mõistetest ning seejärel liigutakse aina keerulisemate teemade juurde. Autor järjestab teemad lisaks sellele, millest järgmises peatükis räägitakse, et eelnevas peatükis õpitut saaks järgmises kasutada. Autor järgib ka eeskujuks võetud õppeaine struktuuri teemade järjestamisel. Ülesannete ja teemade keerukust hindab autor ka oma kogemusest lähtuvalt ning täpsema tagasiside saab õppematerjali testides.

Koodinäited ei ole lisatud lisadesse, vaid otse peatükkidesse kus neid käsitletakse, et õppuril oleks mugavam õppematerjali kasutada ja ta ei peaks pidevalt õppematerjali kasutades edasi-tagasi navigeerima. See tagab sujuva ning mugava lugemise. Koodinäidetes kasutatakse samasugust kirjastiili ning värvust nagu päris koodiski, et õppuril oleks lihtsam jälgida juhendeid ja tekiks vähem segadust. Koodinäidete värvus võib olla erinev tulenevalt kasutatavast ekraanist.

Õppematerjalis kasutatakse oma loodud objektide, klasside,meetodite, atribuutide, muutujate ja funktsioonide puhul eestikeelseid nimetusi, et õpilasel, kes materjali loeb oleks arusaadavam, millised meetodid ja objektid on tema poolt loodud ning millised kaasnevad Objective-C, Foundation ja Cocoa raamistikuga.

Tähtsamad mõisted ning vihjed on ümbritsetud autori poolt kastiga, millel on must piirjoon, et õppurile need koheselt silma paistaks ja ta kindlasti neid loeks. Samuti on seletava ja põhjendava teksti sees olevad Objective-C keelega seotud teksti osad *bold* stiilis, et need tavalisest tekstist eristatavad oleks.

## 3.2 Sihtrühm

Selles peatükis kirjeldab autor milliseid teadmisi ja oskusi, peaksid õppematerjali kasutajad omama, et õppematerjali edukalt kasutada ja määratleb sellega ka sihtrühma, kellele õppematerjal loodud on.

Sihtrühma kuuluvad õppurid peaks teadma ja oskama kasutada järgnevat:

- Tunneb programmeerimisega seonduvaid mõisteid.
- Oskab programmikoodi töötlemiseks kasutada sobivat keskkonda, programmikoodi siluda ja testida.
- Teab ja tunneb objekt-orienteeritud programmeerimise põhimõisteid ning oskab neid programme luues kasutada.
- Teadma viitamisega seotud mõisteid ja oskama neid kasutada.
- Teadma algoritmide ja andmestruktuuridega seonduvaid mõisteid.

- Teadma, mis on MVC(ingl.k Model-View-Controller) arendus muster ning oskab vastavalt sellele programme luua.

Kindlasti tuleb kasuks sihtrühma kuuluval õppuril C keelega programmeerimise kogemus, kuid see ei ole nõutud. Objective-C keel pärineb C keelest ning C keelt saab ka Cocoa raamistikuga arendusi luues kasutada.

Õppematerjali sihtrühmaks sobiksid näiteks Tallinna Ülikooli informaatika bakalaureuseõppe tudengid, kes on läbinud järgnevad ained:

- Programmeerimise alused
- Algoritmid ja andmestruktuurid
- Programmeerimise põhikursus

Nimetatud ained aitavad ja annavad vajalikud baasteadmised autori loodud õppematerjali läbimiseks ning selle abil on autoril võimalik võtta arvesse, millised teadmised on õppematerjali kasutaval tudengil/õppuril olemas ja autor saab arvestada sellega, et vastavaid teadmiseid ei pea hakkama õppematerjalis põhjalikult tutvustama/õpetama, kuid samas võib neid lühidalt korrata.

## 3.3 Õppematerjali struktuur

Materjal on jagatud 6 õppetükiks mis omakorda jagunevad alampeatükkideks, milles antakse esmased vajalikud teadmised, et Mac OS X platvormile lihtsamate rakenduste loomisega toime tulla, kasutades selleks Cocoa raamistikku ja Xcode'i arenduskeskkonda.

Esimese peatükis tutvustatakse Xcode 5.0 arenduskeskkonda ja Xcode Profiler Instumenti'i, et õppur teaks, mida lisaks arendusvahenditele lisaks arenduskeskkond sisaldab. Autor tutvustab lühidalt Cocoa raamistikku ning Foundation raamistikku, mis koosnevad erinevatest teekidest, mis sisaldavad primitiivsete objektide klasse, mida saab kasutada oma rakenduse loomisel. Vastavaid raamistikke tutvustab autor seetõttu, et õppuril oleks teada, millega ta oma rakendusi täpsemalt arendama hakkab. Autor tutvustab ARC'i. ARC'i tutvustamine on seetõttu tähtis, et õppurid oskaksid ARC'i oma programmides kasutada ja läbi selle arendamise protsessi lihtsamaks ning kindlamaks muuta.

Teises peatükis luuakse esimene lihtsam programm, mille abil näidatakse, kuidas aknal olevat kasutajaliidese elementi mõne sündmuse/tegevusega siduda ja kuidas uut projekti tekitada.

Samuti tutvustatakse Xcode arenduskeskkonna osasid, mida käsitletakse õppematerjali kasutades. Peatükk annab esimesed oskused sündmuste ning kasutajaliidese elementide ning nende meetodite sidumisest.

Kolmandas peatükis räägib autor Objective-c süntaksist ja eripäradest ning toob välja ka teatud erinevused võrreldes teiste tuntumate kõrgtaseme programmeerimis keeltega nagu C# või Java. Tuuakse näited meetoditest, tsüklitest, objektidest ja klassidest. Peatükk on vajalik selleks, et viia õppur kurssi Objective-C süntaksi ja tema eripäradega võrreldes teiste tuntumate kõrgtaseme keeltega.

Neljandas peatükis käsitlen olulisi kasutajaliidese elemente. Tabelid, rippmenüüd, tekstiväljad ja nupud on enim kasutatud kasutajaliidese elemente, mida programmeerijad/arendajad oma rakenduste loomisel kasutavad. Vastavate elementide abil suudab õppur informatsiooni kasutajale kuvada ja kasutaja poolt sisestatud informatsiooni töödelda ning kasutaja poolt sooritatud toimingu korral mõne sündmuse käivitada Xcode arenduskeskkonda ja käsitletud arendusvahendeid kasutades.

Viiendas peatükis räägitakse andmete salvestamise võimalustest ning tuuakse näide SQLite'i kasutades. Peatükis õpitakse ka välist teeki kasutama oma projektis. Välist teeki kasutatakse täiendava funktsionaalsuse lisamiseks. Samuti toob autor näite ühe enim kasutatud andmebaasiga ja nagu teada salvestavad programmid andmeid ning üheks võimaluseks on andmete salvestamine andmebaasi. Õpetatakse kasutama ka SQLite Manager'i lihtsamaid komponente, et ülesande sooritamisel oleks võimalik andmebaasi hallata. SQLite Manager'i valis autor teiste sarnaste tarkvarade seast seetõttu, et see on tasuta tarkvara ja autor on eelnevalt sellega kokku puutunud ning ülesannete lahendamiseks sisaldas see kõiki vajalike funktsionaalsusi.
Välise FMDB teegi valiku põhjuseks oli samuti eelnev kogemus. Autor ei kasutanud ainult Xcode'is olemas olevat teeki andmebaasi päringute sooritamiseks, kuna ta soovis näidata juhendi käigus, kuidas mõnda välist teeki oma projekti lisada. Teegi lisamine projekti näitas ka seda, et teekide abil on võimalik oma arendustööd kiiremaks muuta. Ei ole vaja päris algusest peale kõike ülesehitada, vaid saab vajaduse korral ka välist teeki kasutada.

Kuuendas peatükis tutvustatakse õppematerjali kasutajale Xcode'i komponent testimise raamistikku ning luuakse lihtsam test. Peatükk annab õppurile teadmised, kuidas luua lihtsamaid komponent teste, et oma rakendust funktsionaalsust testida Xcode'i arenduskeskkonnas.

## 3.4 Õpiväljundid

Selles peatükis kirjeldab autor, millised on oodatavad õpitulemused õppematerjali läbinud õppurile. Samuti räägib autor edasiõppimise võimalusest ja sellest, kuidas võiks õppematerjali tulevikus kasutada.

Õppematerjali läbinu peab oskama kasutada järgnevaid kasutajaliidese elemente:

- Tabelit
- Rippmenüüd
- Tekstivälja
- Nuppu
- Silti

Õppematerjali läbinu peaks oskama järgnevat:

- FMDB teegi abil sooritada lihtsamaid päringuid andmebaasist.
- Oskab kasutada massiive.
- Oskab kasutada tsükleid, meetodeid, klasse ja objekte Objective-c keeles.
- Teab, mis vahe on NSMutableArray'il ja NSArray'il.
- Oskab luua XCTest raamistikku kasutades lihtsamaid teste.
- Teab, mis on ARC ja kuidas ta aitab kaasa arenduste rakendamisele.
- Suudab lisada Xcode'is olemasolevaid kui ka väliseid raamistikke ja teeke enda projekti.

Õppematerjali tuleks tulevikus täiendada, sest õppematerjal andis lühikese ülevaate võimalustest ja teadmistest Mac OS X'ile arendamisest Objective-C keelt, Cocoa ja Foundation raamistikke ja Xcode'i arenduskeskkonda kasutades.

Edasiõppimiseks kahjuks eestikeelset materjali ei ole ja enda täiendamiseks tuleks ikkagi inglisekeelseid materjale kasutada. Autor soovitab materjali läbinutel edasiõppimiseks lugeda 1.1 alampeatükis nimetatud materjale.

Õppematerjalis käsitletavad teadmised on universaalsed ja tulevad kasuks ka iOS'ile rakendusi arendades, sest nagu Mac OS X'ile luuakse ka iOS'ile rakendusi Objective-C keeles. Küll aga on Cocoa ja Cocoa Touch raamistikud erinevad, kuid Objective-C keele tundmine ning oskus

kasutajaliidese objekte siduda oma koodiga tuleb kasuks ka Cocoa Touch raamistikuga rakendusi luues.

# Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks on õppematerjali koostamine Mac OS X platvormile töölaua rakenduste loomiseks, kasutades selleks Cocoa raamistikku ja Xcode arenduskeskkonda ning tuua välja erinevusi Mac OS X ja Windows'i platvormil arendamisest. Seetõttu, et eestikeelset materjali Mac OS X platvormile töölaua rakenduste loomisest Cocoa ja Xcode'iga ei leidnud, otsustas autor koostada õppematerjali Mac OS X platvormile töölaua rakenduste arendamisest. Samuti oli õppematerjali loomise ajendiks põhjus, et autor ise töötab vastavale platvormile arendajana. Sellest tulenevalt soovis autor anda edasi ka oma teadmisi vastavale platvormile töölaua rakenduste arendamisest.

Bakalaureusetöö käigus valmis õppematerjal, milles autor käsitles lühidalt, kuidas luua erinevaid Mac OS X platvormile omaseid vahendeid kasutades töölauarakendust.

Autori jaoks valmistas õppematerjali koostamisel kõige rohkem raskusi ülesannete ja juhendite koostamine selliselt, et need oleksid lugejale arusaadavad ja seletaksid piisavalt tehniliselt, kuid mitte liiga keeruliselt materjalis käsitletavad teemad lugeja jaoks lahti.

Samuti tegi õppematerjali koostamise protsessi keerulisemaks paljude Cocoa raamistikku ja Objective-C keelde kuuluvate eestikeelsete mõistete puudumine.

Kokkuvõttes võib autor väita, et koostas õppematerjali, mille abil võiksid õppurid omandada esmased teadmised Objective-C keelest, Xcode'i arenduskeskkonnast ja Cocoa raamistikest just Mac OS X'ile töölaua rakendusi valmistades. Koostatud materjali võiks kasutada alternatiivina mõne Windows'i platvormile keskenduva aine asemel.

# Summary

The purpose of this bachelor thesis is to create an material for Mac OS X desktop application's development with Cocoa framework and Xcode integrated development environment and also to bring out the differences in desktop applications development for Windows and Mac OS X platform using .Net framework and C# on Windows and Objective-C and Cocoa framework on Mac OS X. Since author did not find any material that is specially concentrated on developing desktop applications for Mac OS X platform in Estonian language, the author decided to develop material for learning how to develop desktop applications on Mac OS X platform.

During the making of the bachelor thesis, the author was able to produce a learning material, that described how to build Mac OS X desktop applications with native tools and languages.

The most difficult part for author, while building the material, was making the exercises and guides so that the material would be easily readable for the reader and the text that explained how things work would be technical, but not very complicated. Also the lack of some technical words translations in to Estonian language, related to Objective-c programming language and Cocoa framework, made the material making process difficult for author.

The author can claim in summary, that he produced an learning material, that can be used to learn and acquire basic knowledge of developing desktop applications for Mac OS X platform with Cocoa framework, Xcode and Objective-C language . The author has also an idea that the learning material could be used as an alternative for a course that teaches similar things that were taught in author's material.

The first chapter gives an overview of English materials about Mac OS X desktop applications development. All those materials have been read and analyzed by author. Author also gives a short overview over the possibilities of learning development for the above mentioned platform in Estonia.

In the second chapter author descirebs more precisecly about the making of the learining material and the tools that are used in the material. Author brings out other ways how to develop applications to Mac OS X platform and tries to give a brief overview about the differences of developing applications on Mac OS X platform with Cocoa framework and on Windows platform with .Net framework.

In the third chapter author explains the architecture of the material and describes the audience whom the material is made for and what kind of knowledge the users of the material should have about software developing before using the material. Author also sets the outcome goals that the learners who use the material should adchive.

# Kasutatud kirjandus


Apple Inc. (2013). *Mac Developer Library* . Kasutamise kuupäev: 20. March 2014. a., allikas Apple Inc.: https://developer.apple.com/library/mac/navigation/index.html

Apple Inc. (2007). *AppleScript Overview*. Kasutamise kuupäev: 22. 04 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/applescript/Conceptual/AppleScriptX/AppleScriptX.html

Apple Inc. (2013). *Sprite Kit Framework Reference*. Kasutamise kuupäev: 22. 04 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/SpriteKit/Reference/SpriteKitFramework_Ref/_index.html

Big Nerd Ranch. (2014). *Objectivec programming the big nerd ranch guide*. Kasutamise kuupäev: 16. 03 2014. a., allikas Big Nerd Ranch: https://www.bignerdranch.com/book/objectivec_programming_the_big_nerd_ranch_guide

*CocoaPods*. (2014). Kasutamise kuupäev: 15. 04 2014. a., allikas http://cocoapods.org/

Hawkins, J. (2011). *Cocoa and Objective-C Cookbook.* Birmingham: Packt Publishing.

Hillegass, A., & Ward, M. (2013). *Objective-C Programming: The Big Nerd Ranch Guide (2nd Edition).* Big Nerd Ranch Guides.

Kippar, J. (2013). *.Net raamistik.* Kasutamise kuupäev: 20. Aprill 2014. a., allikas Kursuseprogramm: http://minitorn.tlu.ee/~jaagup/kool/java/kursused/13/dotnet/juht.html

Outercurve Foundation . (2014). Kasutamise kuupäev: 01. 05 2014. a., allikas NuGet: http://www.nuget.org/

Poola, K. (6. September 2010. a.). Sissejuhatus Apple-i mobiilsetele tehnoloogiatele. Kasutamise kuupäev: 14. Aprill 2014. a., allikas Õppekava aine: https://itcollege.ois.ee/et/curriculum-subject/view?curriculum_id=3&subject_id=213&year=2013

Villems, A., Kusmin, M., Peets, M.-L., Plank, T., Puusaar, M., Pilt, L., et al. (2012). *Juhend kvaliteetse õpiobjekti loomiseks.* Tallinn: Eesti Infotehnoloogia Sihtasutus.

# Lisad

**Lisa 1 Õppematerjal**

**Lisa 1 Õppematerjal**

Tallinna Ülikool
Informaatika Instituut

# Õppematerjal töölauarakenduste programmeerimisest Mac OS X platvormile

**Lauri Roomere**

**Tallinn 2014**

## Sissejuhatus

Käesolev õppematerjal käsitleb töölauarakenduste loomist Mac OS X platvormile kasutades selleks peamiselt Xcode 5.0 arenduskeskkonda,Objective-C keelt ja Cocoa raamistikku. Peatükis 5 on kasutatud ka välist teeki FMDB, mis lihtsustab SQL'i kasutamist ja SQLite Manageri andmebaasi haldamiseks.

Õppematerjali eesmärgiks on anda lühike ülevaade sellest, kuidas valmistada rakendusi Mac OS X platvormile.Samuti sisaldab õppematerjal näiteid sellest, kuidas oma rakendust testida ja väliseid teeke linkida oma rakendusega.

Õppematerjal on loodud huvilistele, kes soovivad iseseisvalt luua rakendusi Mac OS X platvormile. Õppematerjali võiks kasutada alternatiivina kursustel, kus tavapäraselt keskendutakse Windows platvormile rakenduste loomisele, et tutvustada ka teisi operatsiooni-süsteeme ning neile omaseid (ingl.k native) arendusvahendeid.

Õppematerjalis luuakse enamikes peatükkides uus rakendus, et oleks lihtsam jälgida juhendit.

Eelteadmised, mida on õppematerjali edukaks kasutamiseks teadma peaks on järgnevad:

- objektorienteeritud programmeerimimine
- viitamine
- mäluhaldamine
- krütpeerimine/dekrüpteerimine ning kokkupuude mõne varasema programmeerimis keelega (kasuks tuleb kindlasti C, C++ või C# keele oskus, vastavate keelte teadmine pole nõutav).

Esimeses peatükis tutvustatakse Xcode 5.0 arenduskeskkonda ja vajaminevaid töövahendeid ning esmaseid oskusi, et vastavas arenduskeskkonnas töötamisega toime tulla. Õpetatakse uut projekti looma. Samuti tutvustatakse milliseid erinevaid projekte peale Cocoa rakenduse on veel võimalik luua ning milliseid Xcode'i komponente kasutada.

Teises peatükis luuakse esimene rakendus, mille abil õpitakse objektide sidumist sündmustega, kasutajaliidese elementidest tutvustatakse nuppu, silti ja tekstivälja.

Kolmandas peatükis tutvustatakse lühidalt Objective-C keele süntaksit. Täpsemalt objekte, klasse, meetodeid, objektide lähtestamist, tsükleid ja sõnastikke. Tutvustatakse ka mõningaid massiivide meetodeid. Autor toob mõningates kohtades välja ka Objective-C süntaksi võrdlusi teiste keeltega.

Neljandas peatükis käsitletakse tüüpilisi kasutajaliidese elemente nagu näiteks tabel ja rippmenüü. Peatükis näidatakse kuidas vastavaid kasutajaliidese elementide tegevusi kuulata ja neisse andmeid saata.

Viiendas peatükis luuakse ühendus andmebaasiga FMDB teegi abil ning tutvustatakse ka lühidalt SQLite Manager'i, mille abil saab andmebaasi hallata. FMDB teegi lisamisega näitatakse ühte viisi kuidas on võimalik teeke oma projekti lisada ning samuti näidatakse kuidas Xcode'i arenduskeskkonnas olemasolevat teeki oma projekti lisada. Näites luuakse rakendus, mis lisab ja küsib andmebaasi andmeid .

Kuuendas peatükis luuakse komponent test, mille abil koodi testitakse ning tutvustatakse lühidalt, kuidas Xcode arenduskeskkonas teste luua ja läbi viia, kasutades selleks XCTest raamistikku.

# Sisukord

## Olulisemad mõisted

*Interface Builder*- tarkvara arendamise rakendus, mis kuulub Xcode'i arenduskeskkonda . Selle abil luuakse rakendustele kasutajaliidest. *Interface Builder* on graafilise kasutajaliidesega ning *Interface Builder*'i failinimede laienduseks on xib (Apple Inc., 2012).

*Outlet –outlet*'iks kutsutakse objekti atribuuti, mis viitab teisele objektile. Näiteks, kui soovite kasutajaliidese elemendi ühendada oma loodud klassiga, et kasutajaliidese elementi vastavast klassist kontrollida ja sellele ligi pääseda. *Outlet*'e luuakse ja ühendatakse *Interface Builder*'is (Apple Inc., 2012).

Delegaat (ingl.k *delegate*) - delegaat on objekt , mis käitub vastavalt või kooskõlas mõne teise objektiga, millega on toimunud programmis sündmus. Delegaat objekt on tihti objekt, mis reageerib mõnele sündmusele (Apple Inc., 2012). Näiteks kui kasutaja vajutab nuppu, teavitatakse kasutajaliidese elemendi poolt vastavat objekti, mis kuulab selle kasutajaliidese elemendi poolt toimuvaid sündmusi, et nupuga on toimunud mingi sündmus.

*Data Source* Xcode'i *Interface Builder*'is- *Data Source* on kasutajaliidese andmete kontrollimise delegaat. Vahendab andmeid teatud meetodite abil objekti ja vaate vahel (Apple Inc., 2012).

Mäluleke (ingl.k *Memory Leak*) - programmid kasutavad operatiivmälu_vastavalt vajadusele ja kui programmeerija on unustanud programmi kirjutada mälu vabastamise, kui seda enam vaja pole, siis seda osa mälust enam muuks otstarbeks kasutada ei saa. Seda samahästi kui pole enam arvutis olemaski ning jääb mulje, nagu oleks osa mälust arvutist "välja lekkinud" (Vallaste, 2014).

Haru (ingl.k thread)- programmeerimises kutsutakse haruks iseseisvalt, teistest programmiosadest sõltumatult täidetavat programmiosa. Hargtöötlust toetavad opsüsteemid võimaldavad programmeerijatel koostada programme eraldi valmiskirjutatud ja testitud harudest (Vallaste, 2014).

# 1 Xcode 5.0 arenduskeskkond

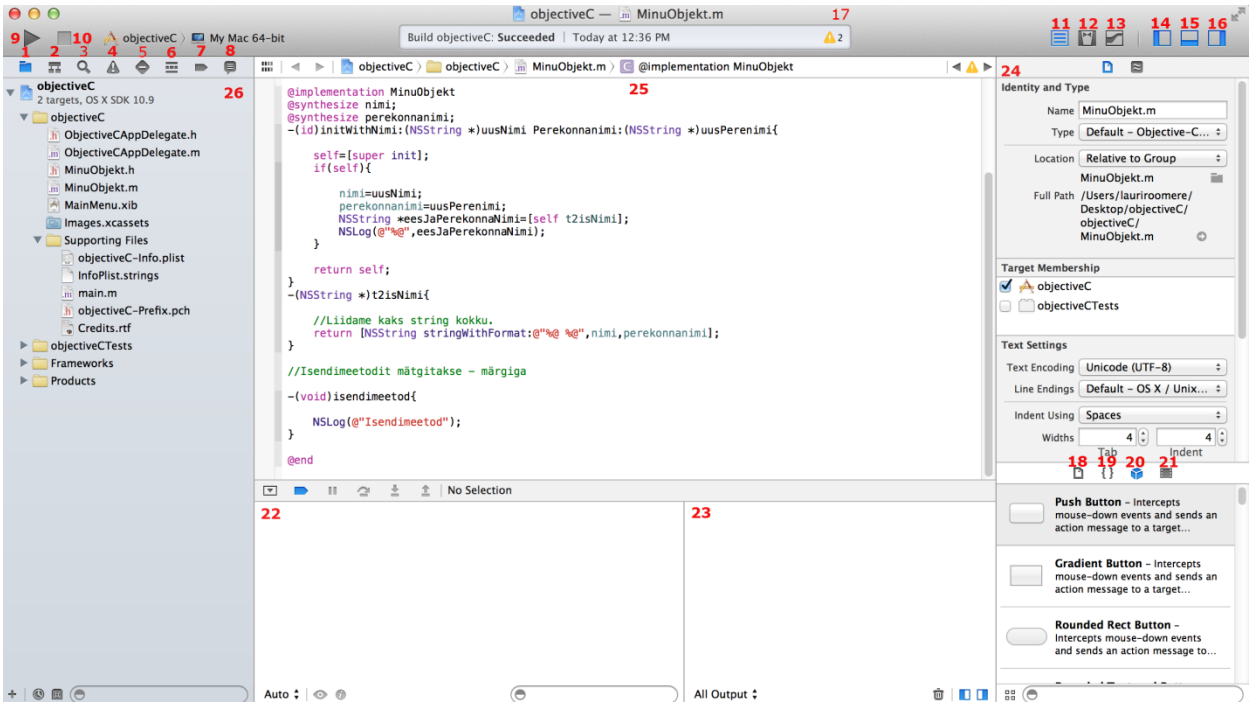
Xcode 5.0 arenduskeskkond (vt Joonis 1) on enim kasutatud arendukeskkondi Mac OS X platvormil. Xcode'i abil on võimalik oma koodi testida erinevate *Profiler Instruments*'i poolt pakutavate tööriistade abil. Komponent testimiseks võib vabalt kaasata mõne teise arenduskeskonnaga ühilduva raamistiku (nt. OCUnit), mille abil oma koodi erinevaid komponente testida või kasutada selleks Xcode'i integreeritud komponenttestimis raamistikku XCTest.

Lisaks sisaldab Xcode Static Analyser'it, mis võimaldab analüüsida teie koodi ning toob välja erinevad vead ning ohud mida analüüsitud kood sisaldab (Apple Inc, 2014). Xcode'i on sisse integreeritud ka Git versiooni halduskontroll, mille abil saab oma koodi hallata.

Xcode on tasuta allalaetav App Store'ist. Xcode 5.0 arenduskeskonnast ning selle paigaldamisest on võimalik lugeda ka Romil Rõbtsenkovi seminaritöös „Rakenduse loomine iOS operatsioonisüsteemiga seadme jaoks“ .

Xcode'is on võimalik luua järgnevalt loetletud rakendusi Mac OS X platvormile:

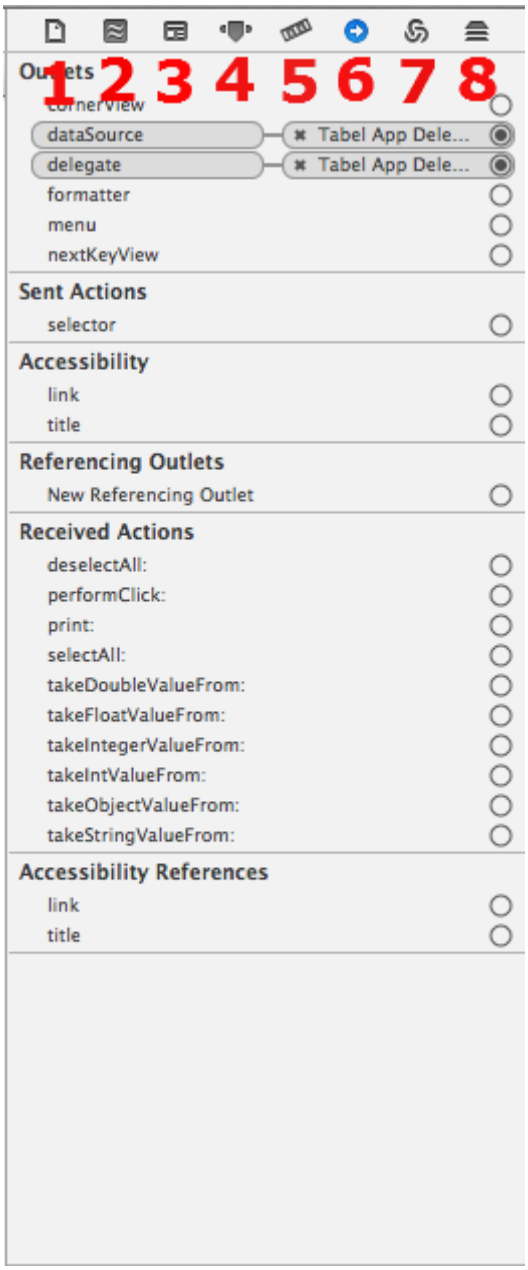
- SpriteKit Game – saab luua 2D mänge kasutades Sprite Kit raamistikku. Sprite Kit raamistiku kasutamiseks on vaja vähemalt Mac OS X 10.9 (Maverick) versiooni. Sprite Kit raamistik sisaldab graafika visualiseerimis ja animeerimis taristut, mida saab kasutada tekstuurpiltide või *Sprite*'ide animeerimiseks (Apple Inc., 2013).
- Cocoa-AppleScript Application – saab luua rakenduse mis on kirjutatud kasutades AppleScript'i. AppleScript on skriptikeel, mis võimaldab otsest kontrolli skriptitavate rakenduste ja paljude Mac OS osadele (Apple Inc., 2007).
- Command Line Tool - võimaldab luua käsurea tööriista.




**Joonis 1. Arenduskeskkona Xcode nummertatud ekraanitõmmis**

1. *Project Navigator*- võimaldab tekitada uusi kaustasid, kustutada kaustasid, grupeerida failisid ja üldiselt neid hallata ja valida faili, et seda näha või redigeerida selle sisu redaktori alas.
2. *Symbol Navigator*- saab vaadelda kõiki projektis olevaid sümboleid või ainult kindlalt määratud sümboleid.
3. *Search Navigator* – otsingu navigaatori abil saab otsida ja filtreerida teksti oma projektidest ning raamistikest.
4. *Issue Navigator*- kuvab probleeme, mis on seotud projektis olevate hoiatuste, vigade, analüüsimise ja projekti järgu (ingl.k *build*) loomisega.
5. *Test Navigator*- võimaldab luua, hallata ja käitada komponentteste ning annab ülevaate testide staatusest.
6. *Debug Navigator*- võimaldab uurida käitatud harusid ja sellega kaasnevate pinude informatsiooni mingil kindlal ajahetkel või momendil programmi täitmise ajal.
7. *Breakpoint Navigator*- võimaldab katkestuspunkte seadistada.
8. *Log navigator*- võimaldab vaadata järkude, käitamiste ja silumiste ajalugu.
9. *Run button*-koodi käitamiseks ja järgu loomiseks mõeldud Run nupp. *Hiire nuppu all hoides on võimalik käivitada järgmiseid protsesse:Run, Test, Profile, Analyze*
10. –*Stop button*- Stop nupu abil on võimalik käimas olevat konstrueerimisprotsessi või silumissessiooni peatada.
11. Standard Editor- avab tavalise redaktori akna.
12. Assistant Editor- avab veel ühe lisa redaktori akna.
13. Version editor- võimaldab sama faili erinevaid versioone võrrelda. Kasulik kui kasutatakse versiooni haldusvahendit nagu näiteks Git.
14. Navigator area-avab navigaatori paneeli.

15. Debug area- avab silumis paneeli.
16. Utility area- avab utiliidi paneeli.
17. File templates- eksemplarid tavalist tüüpi failidele ja koodi konstruktoritele. Eksemplari kasutamiseks tuleb see lohistada teegist projekti navigaatorisse.
18. Code snippets - sisaldab lühikesi koodi osasid tarkvara loomiseks. Kasutamiseks lohistage soovitud koodi osa enda lähtekoodi.
19. Objects - sisaldab kasutajaliidese elemente. Kasutamiseks lohistage kasutajaliidese element enda avatud xib faili Interface Builder'i redaktoris.
20. Media Files-sisaldab graafikat,ikoone ja heli faile. Kasutamiseks lohistage kasutajaliidese element enda avatud xib faili Interface Builder'i redaktoris.
21. Katkestuspunkti konsool.
22. Silumis konsool.
23. Utiliitide ala.
24. Redaktori ala.
25. Navigaatori ala.



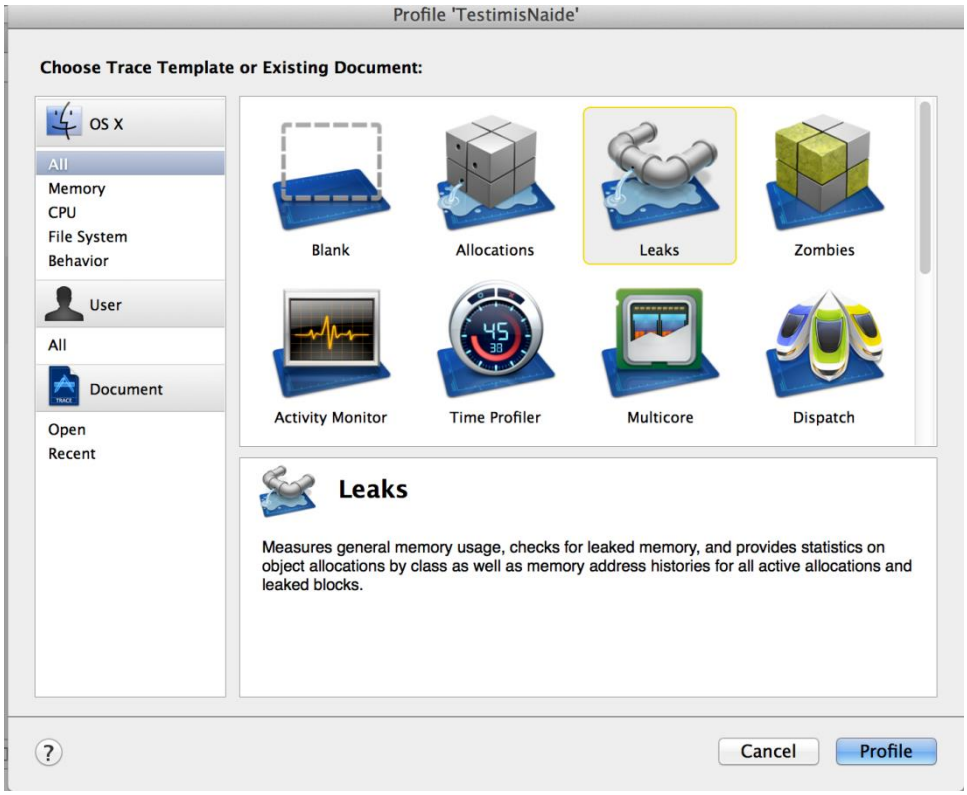

Joonis 2. Utiliitide ala

Utiliitide ala sisaldab lisaks Interface Builderi redaktoris järgnevaid komponente (vt Joonis 2):

1. File inspector- võimaldab vaadelda ja hallata failide metaandmeid. Näiteks nime, tüüpi, teed, asukohta projektis jne.
2. Quick help- saab vaadata detailset informatsiooni mingi sümboli,elemendi kohta (nt mõne kasutajaliidese elemendi), sisaldab kokkuvõtliku kirjeldust sümbolist, kus ja kuidas see on deklareeritd, parameetrid mida ta sisaldab ning selle platvormilisi ja arhitektruuilisi võimalusi.
3. Indentity inspector - saab vaadelda ja hallata elementide metaandmeid, nagu klassi nime, ligipääsu informatsiooni, käitusfaasi atribuute, silti jne.
4. Attributes inspector - saab seadistada valitud kasutajaliidese elemendi atribuute. Nähtavad atribuudid on valitud elemendile omased. Näiteks mõned tekstivälja atribuudid hõlmavad endas teksti joondust ja värvi, piirjoone tüüpi ja muudetavust.

5. Size inspector - võimaldab määrata kasutajaliidese elementide omadusi nagu näiteks elemendi esialgset positisiooni, miinimum ja maksimum suurust ja määrata automaatselt kasutajaliidese elemendi suuruse muutmisega seonduvad reeglid kasutajaliidese elemendile.
6. Connections inspector - võimaldab vaadata outlet'e ja kasutajaliidese elemendi tegevusi, luua uusi ühendusi, eemaldad olemasolevaid ühendusi.
7. Binding inspector - saab luua, vaadelda ja seadistada seoseid vaate elementidele.
8. Effects inspector - saab määrata seadistust animeerimise, alfakanali, üleminekuprotsseside (ingl.k transitions) ja teisi selekteeritud elemendi visuaalseid omadusi.

## 1.1 Xcode Profiler Instruments




Joonis 3. Profiler Instruments'i ekraanitõmmis

Xcode Instruments on jõudluse, analüüsimise ja koodi testimise, dünaamilise jälgimise ning profileerimise vahend.

Vastavate vahendite abil on võimalik jälgida erinevate protsessidega kaasnevaid aspekte ning käitumisi. Näiteks sisaldab Xcode Instruments tööriista nimega *Memory Leaks* mille abil on võimalik tuvastada kus ning millal rakenduses on tekkinud mäluleke. Vastavat tööriista käitades leiate mälulekke allika või võimaliku lekke koha ülesse. Lisaks saab mälu haldamist jälgida vahendit *Allocations* kasutades, mis jälgib mälu kasutamist teie rakenduse poolt. Palju mälust on

kasutuses, milline koodi osa kasutab kõige rohkem mälu, millised objektid teie koodist on hävitatud ning millised veel elavad.

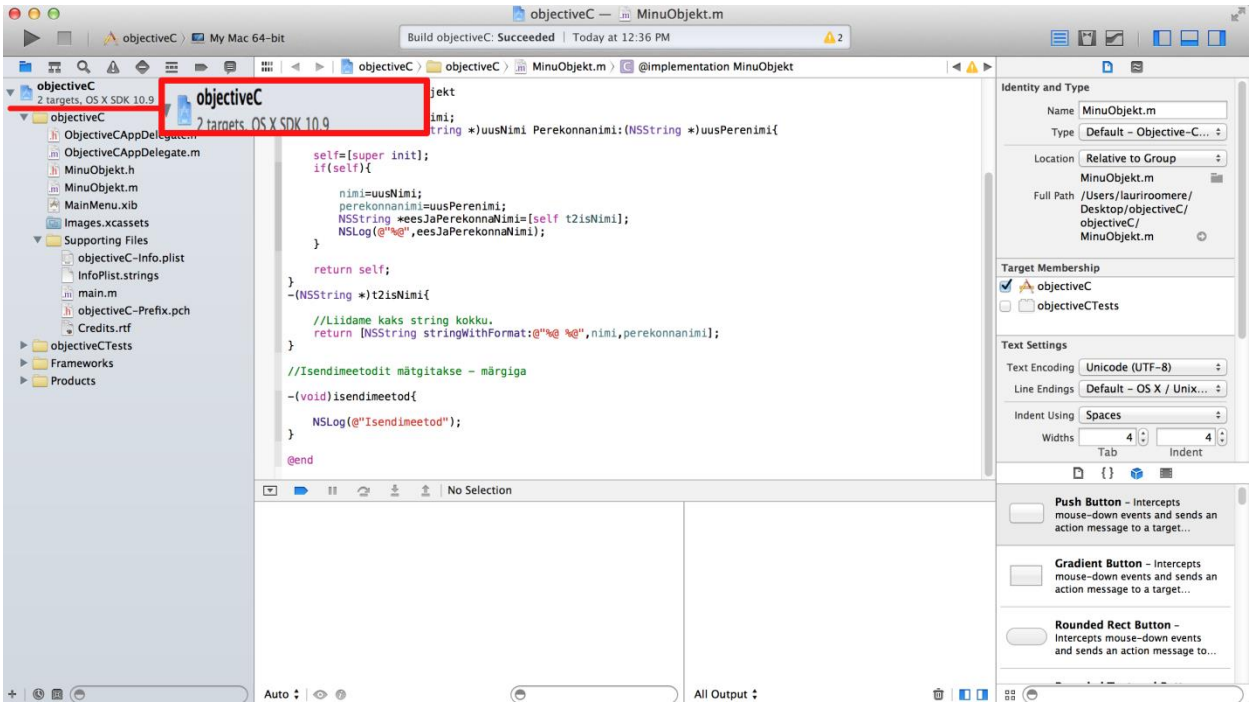
Vastavad vahendid on Objective-C keeles programmeerides väga olulised, sest vaevalt keegi sooviks väljastada oma kliendile rakenduse, mis iga teatud aja tagant kokku jookseb või väga aeglaselt töötab ning on lubamatu väljastada vigaseid, nõuetele mittevastavaid rakendusi. Seetõttu soovitan teil alati oma koodi ja rakendust testida, sest mida hilisemaks vastav tegevus jätta, seda keerulisemaks võivad muutuda probleemid ning lahendused neile.
Rakenduste loomisel kasutan vastavaid vahendeid kui olen mõne uue funktsionaalsuse oma rakendusele juurde lisanud, et näha kuidas see mõjutab rakenduse jõudlust ja töökindlust ning oleks vähem segadust ning keerukust vea leidmiseks hillisemas töö faasis. Mida rohkem ja tihedamalt oma rakendust testida, seda parem.

Heaks inglise keelseks materjaliks, mis juhendab kuidas Profiler Instuments'i kasutada mälu-lekete tuvastamiseks on leitav järgnevalt aadressilt: http://www.raywenderlich.com/2696/instruments-tutorial-for-ios-how-to-debug-memory-leaks

## 1.1 Automatic Reference Counting ehk ARC

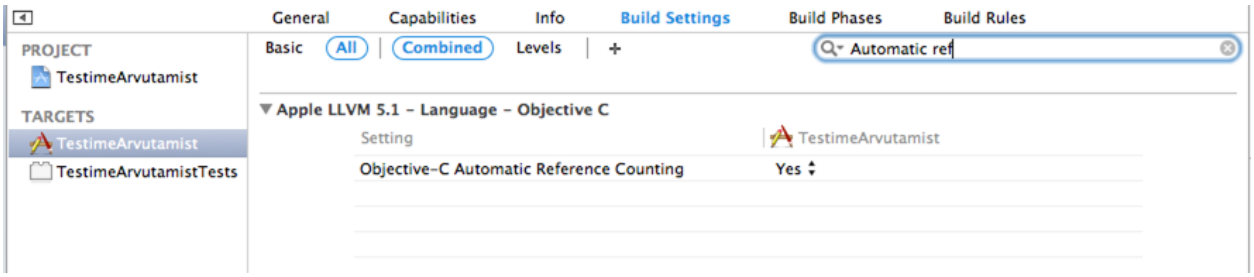
ARC on kompilaatori vahend, mis tegeleb mälu haldamisega. ARC järgib samasid mälu haldamise konventsioone nagu seda tehakse manuaalsel mälu haldamisel. ARC lisab kompileerimise ajal koodi, mille abil ta tagab selle, et objektid oleksid mälus seni kaua kui vaja (Apple inc, 2013).

ARC'i kasutame me ka õppematerjali juhendites ning ülesandeid sooritades soovitab autor ARC'i kasutada. ARC lihtsustab rakenduste loomist Mac OS X platvormile ja aitab õppuril keskenduda juhendi järgimisele ning seetõttu, et autor ei käsitle õppematerjalis manuaalset mälu haldamist, vaid lihtsalt tutvustab seda lühidalt.
Järgnevalt näitab autor kuidas ARC'i oma rakenduses seadistada, täpsemalt siis oma rakenduses võimaldada (ingl.k enable) ning tutvustab *Strong* ja *Weak* tüüpi viitasid.
Mälu haldamist saab võimaldada projekti seadetest. Projekti seadete nägemiseks klõpsame projekti navigaatoris olevale projekti ikoonile (vt Joonis 4 ).




Joonis 4. Projekti seadete avamine

Järgmisena valime redaktori aknas avanenud projekti seadete alt Targets ja paremast menüüst Build Settings. Seejärel sisestame otsinguribasse "Automatic Reference Counting" ja valime jah, et rakenduse loomisel ARC'i kasutada (vt Joonis 5).




Joonis 5. ARC'i seadistamine

Samas ei soovita ma täielikult ARC'i peale lootma jääda, pigem tuleks ARC'i võtta kui abivahendit, mis lihtsustab töö tegemist, kuid mälu haldamist tuleks ikkagi kontrollida. Näiteks ei suuda ARC lahendada järgnevat probleemi, kui kaks objekti viitavad üksteisele Strong tüüpi viitega. Selle tagajärjel tekib *Retain Cycle*, mis ei luba objekti mälust vabastada. Probleemi lahendamiseks tuleks ühele objektile viidata Weak tüüpi viitega. Kui te kasutate ARC'i ja soovite kaasata mõne teegi või koodi osa, mis ei kasuta ARC'i siis selle jaoks tuleb projekti seadete alt vastavale failile lisada käsklus **–fno-objc-arc**.

Vastav käsklus ütleb kompilaatorile, et neis failides ei kasutata ARC'i. Valida tuleks *Targets*, sealt alt oma projekt ja *Build Phases->Compile* Sources topelt klikkida *Compiler Flags* reale ning avanenud aknasse kirjutage eelpool toodud käsklus.

ARC'i abil on lihtsam hallata ka erinevaid viidete tüüpe atribuutidele omistades. Kui viite tüüpi ei ole ise määratud, määratakse atribuudile ARC'i kasutades selleks kohe strong tüüpi viide. ARC vähendab mälulekete võimaluse tekkimist.

Peamised viite tüübid ARC'i kasutades on järgmised:

- *Strong*-objekt püsib elus niikaua kuni sellel on Strong tüüpi viitaja.(Apple inc,2013)
- *Weak*-määrab objektile viite mis ei hoia viidatud objekti elus. Hoitakse nii kaua elus kuni mujalt ei viidata talle Strong viitega (Apple inc,2013).
- *Assign*- kasutatakse C tüüpi muutujate korral.
- *Nonatomic*-mitmel harul on ligipääs muutujale samaaegselt.
- *Atomic*-ainult ühel harul on ligipääs muutujale.

## 1.2 Cocoa ja Foundation raamistik tutvustus

Cocoa raamistik koosneb teekidest, API'dest ja käitusfaasidest(ingl.k runtime) mis moodustavad arenduskihi (ingl.k developmet layer) kogu OS X'ile. Cocoa raamistiku abil rakendusi luues arendatakse rakendusi välja samamoodi nagu OS X'i on arendatud. Rakendused pärivad OS X'i funktsionaalsuse ja välimuse/kasutajaliides, samuti täieliku ligipääsu UNIX'i operatsiooni süsteemil. Cocoa raamistik kasutab MVC mudelit (Apple Inc., 2014). Cocoa raamistik on võrreldav Windows'i arendajatele tuntud .Net raamistikuga.

Foundation raamistik sisaldab põhiklasse, mille abil rakendusi valmistada ning tagab objekt-orienteerituse toe erinevat tüüpi objektidele. Raamistik kasutab eesliidet NS (tähendab NeXTSTEP, või NeXT/Sun)**Invalid source specified.** (vt koodinäide 1).

```
NSString *helloString=@"Tere, see on minu esimene programm Mac OS X'ile";
```

**Koodinäide 1. Näide NS eesliitest**

# 2 Uue projekti loomine

Selles peatükis loome rakenduse, mis kuvab kasutajale nuppu vajutades aknale lohistatud sildi sisse (ingl.k *Label*) „Tere, see on minu esimene programm Mac OS X'ile". Selle väikse rakenduse loomise käigus tutvustatakse, kuidas siduda aknal olev kasutajaliidese objekt mõne sündmuse/tegevusega ja kuidas uut Cocoa rakenduse projekti luua. Samuti puutub õppur kokku **NSLog** meetodiga, mille abil kuvatakse konsooli aknasse teksti ja **NSString** tüüpi objektiga.

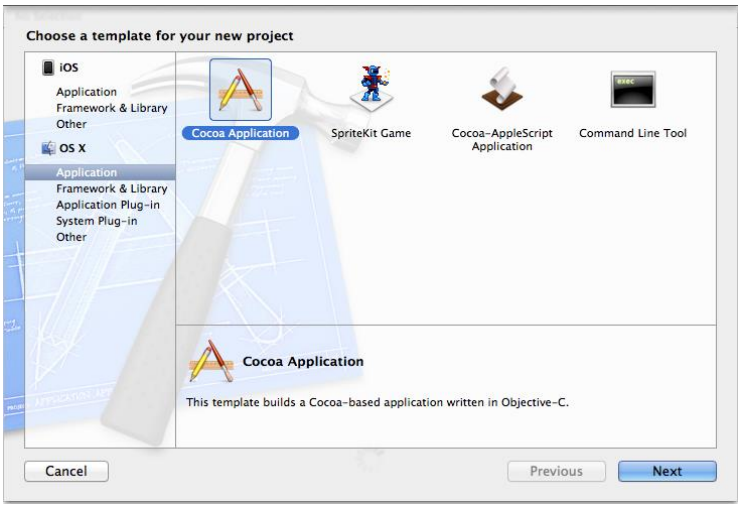
## 1.1 Uue Cocoa rakenduse projekti loomine

Uue projekti loomiseks avage Xcode'i arenduskeskkond ning avanenud aknas valige (Joonis 6. Projekti loomine.*)* ***Create a new Xcode project.*** Kui eelnevalt on mõni projekt ebakorrektselt suletud või mõni muu olemasolev projekt on avatud, siis valige Xcode'i menüüst ***File->New Project.***


Joonis 6. Projekti loomine.

Seejärel avaneb uus aken, kus palutakse kasutajal täpsustada, mis tüüpi rakendust ta soovib luua. Kuna meie soovime luua rakendust Mac OS X platvormile, valime me OS X valiku alt Application ning Create New Cocoa Application, sest õppematerjal on keskendunud just Cocoa'ga rakenduste valmistamisest Mac OS X platvormile. Seejärel vajutage nupule next, et

kinnitada oma poolt tehtud valikut (vt joonis 7).

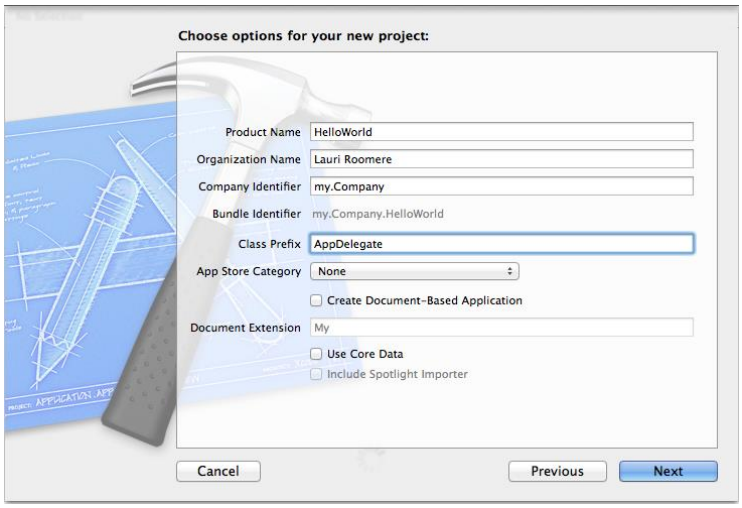



**Joonis 7. Programmi tüübi valimine projekti jaoks**

Pärast nupule vajutust avaneb järgmine aken, kus saab oma loodud projekti seadistada. Seadistuses on võimalik määrata järgnevaid seadeid:

- rakenduse nimi (sellest saab ka projekti nimi), võib lisada ükskõik millise nime (ingl.k *product name*).
- organisatsiooni nimi, valikuline võimalus, lisab organisatsiooni nime kõikide failide algusesse, et oleks teada kelle poolt rakendus loodud on (ingl.k *organization name*).
- Firma, asutuse tunnus, mille järgi saab rakendust tuvastada (ingl.k *company identifier*).
- Klassi eesliide (määratakse klassile, mis kuulab rakenduse peamiseid sündmusi ning millele järgneb pärast eesliidet AppDelegate, mis näitab et tegu on rakenduse delegaate kasutava klassiga) (ingl.k class prefix).
- Kategooria, mille järgi oma toodet App Store'is pakkuda (ingl.k *app store category*).
- Dokumendi laiend, juhul kui loote dokumentidega tegeleva programmi ja soovite ka rakenduses loodut salvestada, lisatakse sellele teiepoolt antud laiend (ingl.k *document extension*).

Sisestage oma projektile/tootele sobiv nimi, määrake põhiklassile eesliide, ülejäänud väljad võib täitmata jätta, ning vajutage next (vt Joonis 8. Projekti ülesseadistamine).




**Joonis 8. Projekti ülesseadistamine**

Xcode keskkonnas on võimalik luua Mac OS X platvormile järgnevaid programme:

- Raamistikke (ingl.k *framework*)
- Teeke (ingl.k *library*)
- Süsteemi lisasid (ingl.k *system plug-in*)
- Rakenduste lisasid (ingl.k *application plug-in*)
- Rakendusi (ingl.k *applications)*

Xcode arenduskeskkonnas on võimalik programmeerida ka C++ ja C keeles töötavaid rakendusi.

Projekti loomisel luuakse projekti nime kandvasse kausta **AppDelegate.m** ja **AppDelegate.h** failid, mida kuvatakse ka projekti navigaatoris, kus on lisaks kaustad toote (ingl.k *Products*), tugi failide (ingl.k *Supporting Files*), raamistike (ingl.k *Frameworks*) ja testide (ingl.k *Tests*) jaoks. Testide kausta ees on projekti nimi, millele järgneb lõppu *Tests*. Vajadusel on võimalik kaustade nimesid muuta ning uusi juurde tekitada, et paremini projekti sisu hallata.

## 1.1 Esimese rakenduse loomine

Kõigepealt avame projekti navigaatorist oma **AppDelegate.m** põhifaili, selle peale klikkides. See fail hakkab sisaldama meie poolt kirjutatud koodi ning fail AppDelegate.h on päisefail. Faile, mis juba loodud on või luuakse kuvatakse kasutajale projekti navigaatoris (vt joonis 9).

**AppDelegate.m** failis on näha juba valmisolevat meetodi osa, mille abil kuulab programm kas rakendus on käivitamise lõpetanud. Tegemist on delegaat meetodiga, mis teavitab rakenduse käivitumisest.

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSLog(@"Rakendus käivitus");
    NSString *helloString=@"Tere, see on minu esimene programm Mac OS X'ile";
    NSLog(@"%@",helloString);
}
```

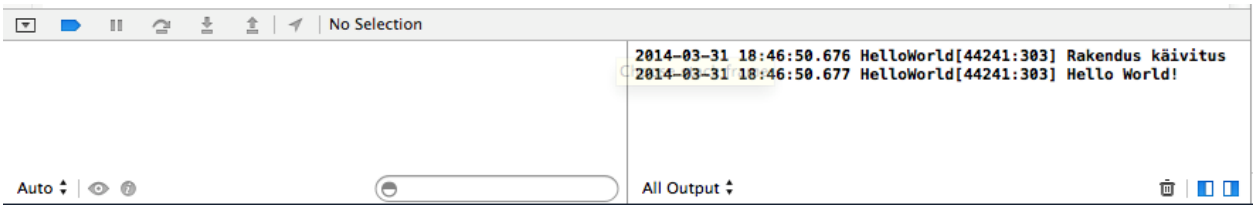
**Koodinäide 2. Rakenduse käivitumist kuulav meetod**

Koodinäide 2 lisame rakenduse käivitumise meetodisse kaks väljastamist, millest üks väljastab sõne tüüpi loodud objekti ning teine lihtsalt meie sõne „Rakendus käivitus".

Selle, mis tüüpi objekti me soovime väljasta määrame **NSLog** meetodi sulgude sees protsendi märgi ning vastava sümboli või tähega (nt **NSNumber** tüüpi objekti korral **NSLog(@"%@")** või int tüüpi muutuja korral **NSLog(@"%i")**), sõnesid luues, on sõne alguses **@** sümbol, millele järgnevad kaks apostroofi **(@"")**.

Kuna me soovime väljastada objekt tüüpi muutujat nimega **NSString** , siis me kasutame **%@** sümboolikat, kõiki **NSObject** klassist pärinevaid objekte tähistatakse printides vastava sümboliga. Kõik Cocoa raamistiku muutujad ning objektid algavad **NS** eesliitega millele järgneb vastava muutuja tüübi nimetus nagu eelpool toodud näites **NSString**. Käivitage rakendus vajutades selleks cmd+r.

Tulemusena peaksite nägema konsooli aknas (vt Joonis 9) välja prinditud sõnu ning teie töölaual avaneb ka rakenduse aken.



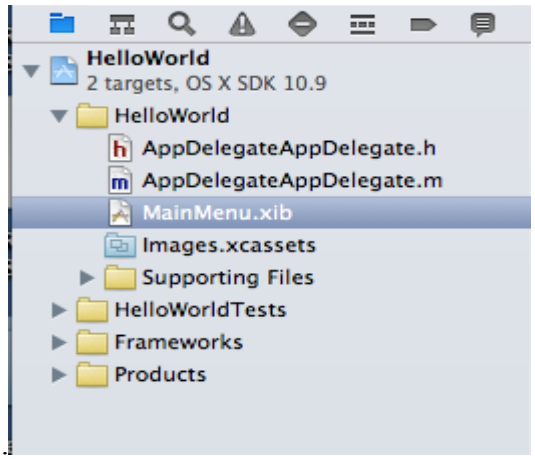

**Joonis 9. Silumis konsooli aken**

Rakenduse peatamiseks vajutage stopp nupule (vt Joonis 1). Kui teil käivitamisel silumis konsool ei avanenud, vajutage tööriista ribal olevale nupule, mis vastavalt seisundile kuvab või peidab silumis konsooli (vt Joonis 10).

**Joonis 10. Silumis konsooli avamine**

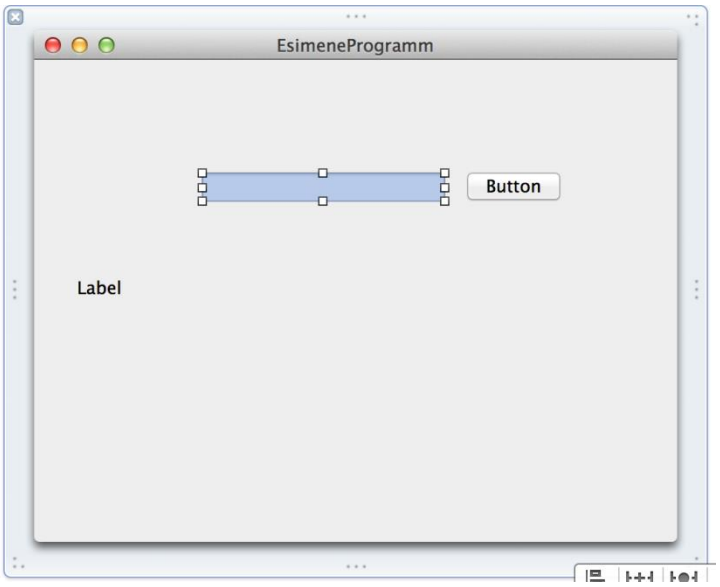
## 2.1 Kasutajaliidese objektide sidumine ja tegevuste kuulamine

Kõigepealt valime projekti navigaatori menüüst (vt Joonis 10) MainMenu.xib faili, mis sisaldab meie rakenduse akent ja disainimiseks vajalikke vahendeid.Failile klikkides avaneb see redaktori alas . Seejärel lisame kaks uut kasutajaliidese eset kasutajaliidese elementide teegist aknale (vt Joonis 11).




Joonis 11. Projekti navigaator

Kui vastav utiliidi paneel ei ole nähtav, tuleks vajutada Joonis 10 kujutatud paremal olevale nupule, mis toob nähtavale utiliidi. Siis valime objekti teegist surunuppu (***push button***) tüüpi kasutajaliidese elemendi ning lohistame selle objekti teegist aknale. Sama protseduuri teeme ka sildi ja tekstiväljaga (vt Joonis 12). **Label**'i puhul on tegu tekstiväljaga, mis sisaldab ja kuvab teksti ning mille tekst on kasutaja poolt selekteeritav.

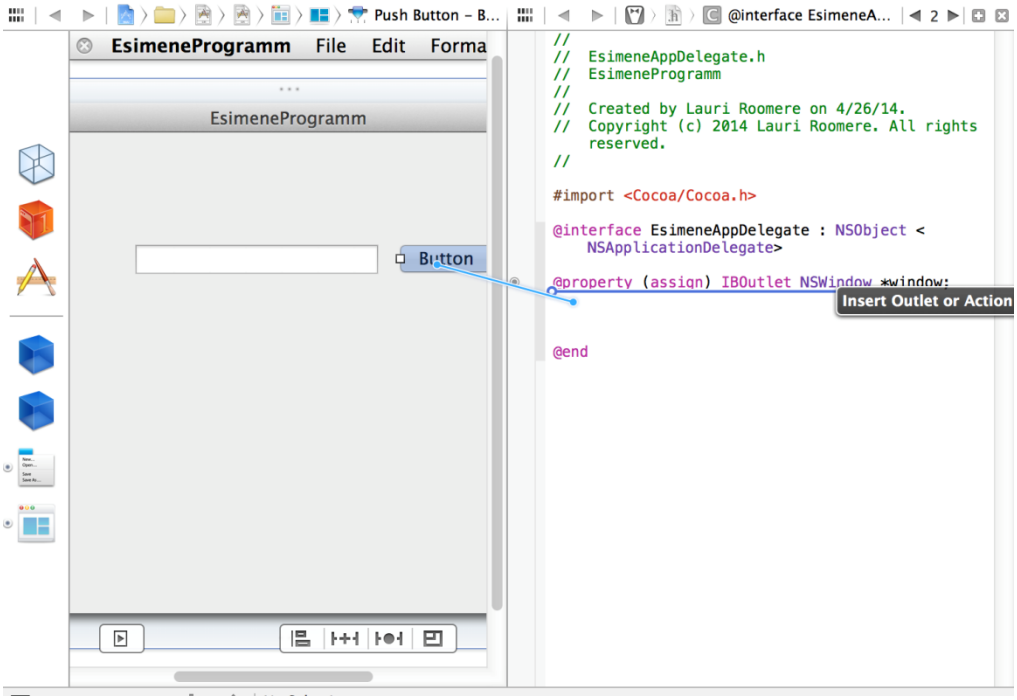



Joonis 12. Sildi ja tekstivälja lisamine aknale.

Järgnevalt ühendame nupu tegevuse oma **AppDelegate** objektiga.Vajutage ***Assistant editor*** nupule , mis asub joonisel 13 keskel.

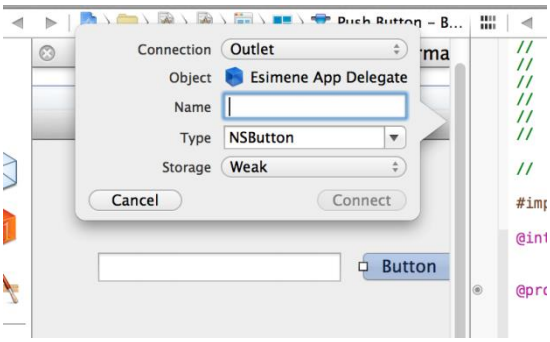
Joonis 13 .Assistant Editor

Nuppu vajutades ilmus meie akent sisaldava vaate kõrvale uus vaade, kus on avatud **AppDelegate.h** fail. Paremal vaates on näha avatud xib faili (vt Joonis 16). Vajutage aknal olevale nupule paremat hiire nuppu all hoides ja lohistage tekkinud joon teises vaates olevasse koodiosasse (vt Joonis 14.).




Joonis 14. Nuppu tegevuse loomise esimene etapp

Pärast seda avaneb uus aken, kus soovitakse teada, mis tüüpi seose me soovime luua ning mis nime me anname oma loodavale objektile (vt Joonis 15).




Joonis 15. Nuppu ühenduse omaduste määramine

Lisaks on võimalik määrata, mis tüüpi viitega on tegu. Seos peaks olema **Action** tüüpi, selle abil saame kuulata kas nupule on vajutatud.

Oma tegevuse nimetan **nupuKuular'iks** ning seejärel vajutan ühenda(ingl.k *connect*). Nüüd on meil olemas ühendus nupuga, mille abil saame kuulata tegevusi. Avame **AppDelegate.m** faili ning näeme, et sinna on loodud meetod nimega **nupuKuular**.

Enne koodi kirjutamise alustamist peame looma veel ühenduse sildi ehk **Label** tüüpi kasutajaliidese elemendi **MainMenu.xib**'is ning kordame sama protseduuri, mille sooritasime nupu ühendamisel, kuid ärge muutke ühenduse tüüpi **Outlet**'ist tegevuseks (ingl.k *action*), sest me soovime selle **Outlet**'i abil lisada nupu vajutuse korral sildi sisse teksti **setStringValue** meetodiga. Sildi puhul ei ole vaja meil tegevusi kuulata, vaid ligipääsu sildi meetoditele.

**AppDelegate.m** faili kirjutame @implementation sektsiooni sisse **AppDelegateAppDelegate** käsu @synthesize **tekstiKast**; mille abil genereerib kompilaator ise juurdepääsumeetodid meie muutujale nimega **tekstiKast**.

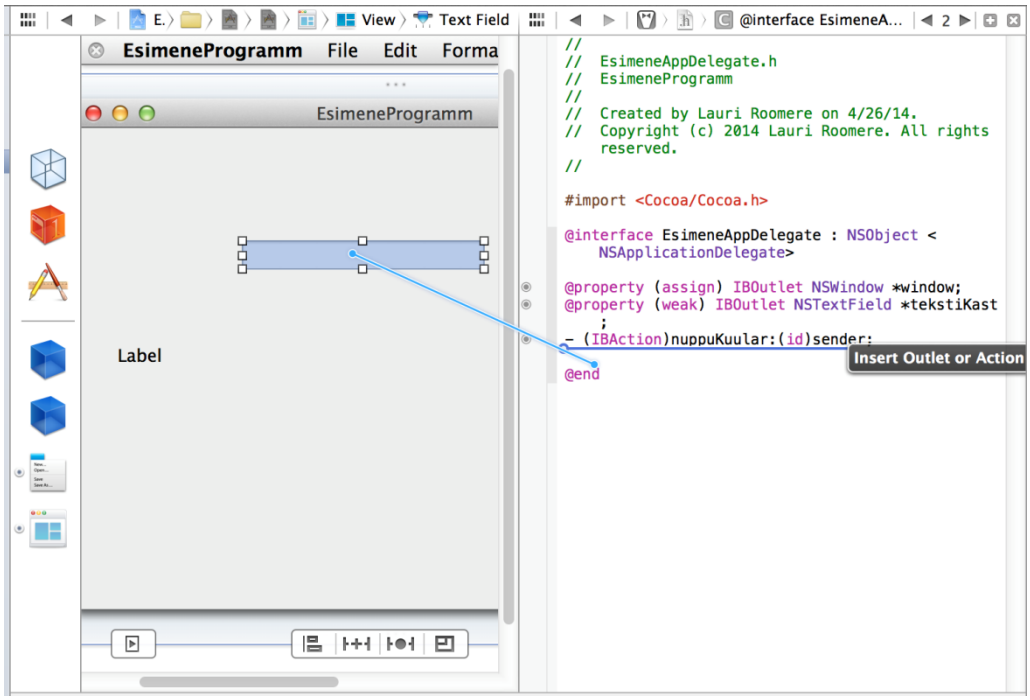
Kui me vastavat toimingut ei teosta, tuleb meil objekti poole pöörduda **self** võtmesõna abil (nt **self.tekstiKast**). **Self** on viit, mis hoiab klassi aadressi mälus (nt C# vastab self'ile this). Siis kutsume **nupuKuularnupuKuular**'i funktsioonis välja **Label** objekti meetodi nimega **setStringValue,** mille abil anname oma objektile string väärtuse, mille kuvame aknal olevasse **Label** objekti nupu vajutusel.(Koodinäide 3. Nupu tegevust kuulav meetod)

```
- (IBAction)nupuKuular:(id)sender {

    [tekstiKast setStringValue:@"Tere, see on minu esimene programm Mac OS X'ile"];
}
@end
```

**Koodinäide 3. Nupu tegevust kuulav meetod**

Programmi käivitamise järel peaks nupule vajutades ilmuma **Label**'i sisse teie poolt sisestatud sõnum. Kui märkate, et kuvatav tekst on liiga pikk, et **Label**'i sisse ära mahtuda, klikkige **Label**'i peale xib failis oleval aknal ja muutke tekkinud abipunktide abil oma **Label**'i suurust nagu seda teisteski keskkondades tehakse.

Nüüd ühendame tekstivälja **outlet**'i **AppDelegate** objektiga ja lisame sildi sisse, tekstivälja kirjutatud teksti, nupu vajutusel (vt Joonis 16).



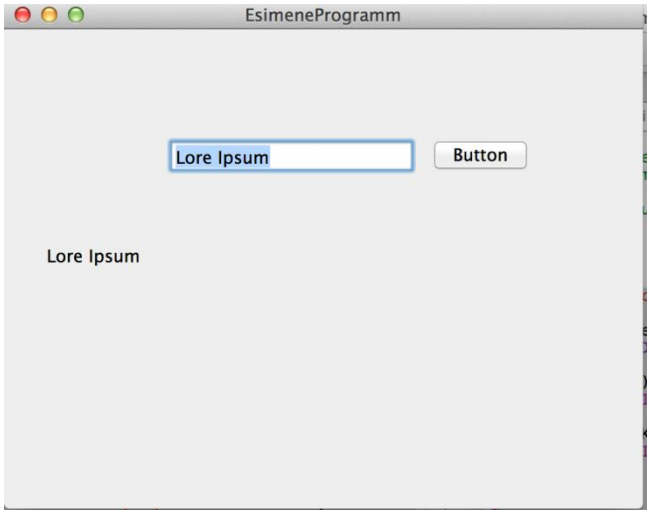

Joonis 16. Outlet'i loomine

Tekstivälja elemendi väärtuse saamiseks, kasutame meetodit **stringValue** (vt Koodinäide 4).

```
- (IBAction)nupuKuular:(id)sender {

    [tekstiKast setStringValue:[tekstiV2li stringValue]];
}// nupuKuular meetodi lõpp
```

Koodinäide 4. nupuKuular meetod

Joonisel 17 on näha selles peatükis loodud rakenduse lõpptulemust.




Joonis 17. Esimese rakenduse tulemus

## Ülesanded

1.Looge uus projekt ning väljastage aknale sildi sisse ning konsooli oma nimi.

2.Lisage aknale nupp ning tekstiväli(**NSTextField**) ja silt(**NSLabel**). Tekstiväljale nime sisestades ning nupule vajutades ilmub silti teie nimi.

# 3 Objective-C

Selles peatükis näidatakse kuidas luua Objective-c keele ja raamistike abil objekte, klasse, meetodeid, tsükleid ja sõnastikke (ingl.k *Dictionary*).

Objective-C on põhiline programmeerimise keel Mac OS X ja iOS platvormile rakenduste loomiseks. Tegemist on kõrgema klassi keelega, mis pärineb C programmeerimiskeelest ning võimaldab kasutada objektorienteeritud programmeerimist ning dünaamilist käitust (ingl.k *dynamic runtime*) (Apple Inc, 2012).

Mac OS X platvormile rakendusi luues kasutatakse suuremalt jaolt objekte, mis on Objective-C klasside eksemplarid (ingl.k *instance*). Rakenduse loomisel kasutatav Cocoa raamistik sisaldab vajalikke klasse rakenduste loomiseks ning vajaduse korral saab ise uusi juurde tekitada. Lisaks on võimalik ka olemasolevaid klasse, mida Cocoa raamistik sisaldab laiendada endale sobivaks klassiks, lisades uusi vajalikke meetodeid klassile juurde (Apple inc, 2012).

## 3.1 Objektid ja klassid

Objective-C keeles tekib uut Objective-C klassi tekitades kaks faili:

- päise fail, mille lõpus on .h faililaiend ning, mis sisaldab **@interface** sektsiooni, kus kirjeldatakse ära meetodid ning muutujad.
- **@implementation** sektsioon faililaiendiga .m, kuhu hakatakse koodi kirjutama ning meetodeid implementeerima.

Samuti kirjeldatakse päise failis alati ära, mis tüüpi klassiga on tegu.
Järgnevalt loetlen ära mõningate klasside tüübid, mida õppematerjalis ka käsitletakse:

- NSObject- on juurklass paljude Objective-C klasside hierarhias. **NSObject**'iga pärivad objektid elementaarsed liidesed käitussüsteemi ja suutelisuse toimida Objective-C objektina **Invalid source specified.**.
- NSString- deklareerib programmilise liidese objektile, mis haldab muutumatut sõne. Muutumatu sõne on tekst, mis on defineeritud objekti tekitamisel ja mida seejärel pole võimalik muuta **Invalid source specified.**.
- NSTextField- **NSTextField** on **NSControl** tüüpi objekti sarnanev objekt, mis kuvab teksti, mida kasutaja saab selekteerida või redigeerida ja mis saadab sündmuse sõnumi oma sihtobjektile kui kasutaja vajutab Return klahvi **Invalid source specified.**.

- NSButton- **NSControl** klassi alamklass, mis kuulab hiire klikkimise sündmusi ja saadab sündmuse sõnumi sihtobjektile, kui seda on vajutatud või klikitud **Invalid source specified.**.
- NSTableView- **NSTableView** objekt kuvab andmeid seotud kogumi kirjetest koos ridadega, mis väljendavad individuaalseid kirjeid ja veergudega mis väljendavad vastavate kirjete atribuute **Invalid source specified.**.

**NSNumber**- NSValue alamklass, mis pakub C tüüpi numbrilisi väärtusi. Defineerib meetodite kogumiku, spetsiaalselt väärtustele ligipääsemiseks ja määramiseks märgitud või märkimata tähemärkidele, lühikestele täisarvudele, täisarvudele, pikkadele täisarvudele, pikkadele pikkadele täisarvudele, ujukomaarvudele, double arvudele ja loogikamuutujatele **Invalid source specified.**.

Objekte on kaht tüüpi, objektid mida saab muuta (nt. NSMutableArray) ja objektid, mida pärast selle loomist muuta ei saa (nt. NSArray), kuhu pannakse kõik objekti lähtestades juba sisse. Võimalik on ka muutumatust objektist teha koopia, millega saab manipuleerida (Uusi objekte juurde lisada, kustuda jne). Muutujad, mis sisaldavad nimes Mutable ehk muudetav nimetust on muudetavad.

Objekte välja kutsudes tuleb objekt lähtestada. Objekti lähtestatakse Objective-c keeles järgnevalt **Objekt *uusObjekt=[[Objekt alloc] init];**. Objekti välja kutsudes eraldame käsuga **alloc** mälu antud objektile.

Kui me ei kasuta ARC'I, siis me peame ütlema millal me soovime objekti mälust vabastada. Iga objekt, mis tekitatakse pannakse mällu ning kui me soovime seda mälust vabastada, kasutatakse selleks käsku **release**.

Lisaks on võimalik kasutada ka **autorelease** käsku, mis vabastab objekti automaatselt mälust kui sellele enam ei viidata strong tüüpi viitega. Õppematerjali järgides ja ülesandeid lahendades kasutame ARC'i , et õppimis protsessi turvalisemaks muuta.

Loome uue projekti, millele anname meie näites nimeks **ObjectiveC** ja eesliiteks samuti **ObjectiveC.** Kui projekt on loodud, loome uue klassi nimega **MinuObjekt**. Lisame kaks atribuuti nimedega **nimi** ja **perekonnanimi** (vt Koodinäide 5).

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

@interface MinuObjekt : NSObject
@property NSString *nimi;
@property NSString *perekonnanimi;
@end
```

**Koodinäide 5. Atribuutide lisamine**

Kõigepealt, et me saaksime loodud klassi **MinuObjekt ObjectiveCAppDelegate**'is välja kutsuda, peame me **MinuObjekt** importima (vt Koodinäide 6. Klassi importimine).

```
#import "ObjectiveCAppDelegate.h"
#import "MinuObjekt.h"
@implementation ObjectiveCAppDelegate
```

**Koodinäide 6. Klassi importimine**

Kutsume loodud klassi **applicationDidFinishLaunching** meetodis välja ja määrame objekti atribuutidele väärtused ning prindime tulemused välja **NSLog()** meetodit kasutades (vt Koodinäide 7. Uue **objekti** loomine.

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    MinuObjekt *minuObjekt=[[MinuObjekt alloc]init];
    minuObjekt.nimi=@"Juku";
    minuObjekt.perekonnanimi=@"Kask";
    NSLog(@"Nimi: %@ Perekonnanimi: %@",minuObjekt.nimi,minuObjekt.perekonnanimi);
}
```

**Koodinäide 7. Uue objekti loomine.**

## 3.2 Meetodid

Kuna Objective-C keele puhul on tegemist objektorienteeritud keelega, räägitakse siin pigem meetoditest kui funktsioonidest. Meetod nagu me teame on seotud objektidega, mis kuuluvad objektile. Seepärast ei räägi me Objective-C keeles meetoditest.

Siiski on alati programmil üks funktsioon olemas, selleks on **main()**. Funktsioon on leitav projekti navigaatoris ***Supporting Files*** kaustas main.m failis.

Objective-c keeles kutsutakse meetodit välja klassis kuhu meetod kuulub järgneval viisil **[self meetod1]** ning mõnest teisest klassist **[objekt meetod1]**.

Isendimeetodit märgitakse Objective-C keeles – märgiga ja klassi meetodeid + märgiga. Klassi meetodit kutsutakse välja **[Klass Meetod]** (vt Koodinäide 8.).

```
    // Klassis A olev kood
-(NSString *)t2isNimi{
    return [NSString stringWithFormat:@"%@ %@",nimi,perekonnanimi];
}
+(void)klassiMeetod{
    NSLog(@"Klassi meetod");
}
```

**Koodinäide 8. Isendimeetodi ja klassi meetodi näide**

Kui paljudes keeltes kasutatakse punkti süntaksit siis Objective-C on meetodid nurksulgudes, kuid teatud juhtumitel on võimalik ka Objective-c keeles punkti süntaksit kasutada. Näiteks atribuudile väärtust määrates (nt**. inimene.nimi=@"Kalle**").

Parameetreid määratakse Objective-C keele meetodis samuti pärast meetodi nime, kuid mitte sulgudesse vaid pärast koolonit **[self meetod1:väärtus parameeter2:väärtus]**.

Meetodile parameetreid kirjeldades antakse parameetrile ka teada, mis tüüpi ta on. Samuti võib kasutada Objective-C raamistikega kaasnevate objektide asemel ka C muutujaid oma koodis. Lisage vastavad meetodid eelnevalt loodud projekti. Vastavad meetodid kutsume välja rakenduse käivitumisel (vt Koodinäide 9. Erinevat tüüpi meetodid.).

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application

    [self meetod1];
    NSLog(@"Teine meetod %@",[self meetod2]);
    NSLog(@"Kolmas meetod %d",[self meetodCTyypiMuutujagaJaParameetriga:2]);
    NSLog(@"Neljas meetod %ld",(long)[self meetodObjcTyypiMuutujagaJaKaheParameetriga:2
parameeter2:3]);

    MinuObjekt *minuObjekt=[[MinuObjekt alloc]init];
    minuObjekt.nimi=@"Juku";
    minuObjekt.perekonnanimi=@"Kask";
    NSLog(@"Nimi: %@ Perekonnanimi: %@",minuObjekt.nimi,minuObjekt.perekonnanimi);
} //applicationDidFinishLaunching meetodi lõpp
-(void)meetod1{

        NSLog(@"Void tüüpi meetod");

}
//
-(NSString *)meetod2{
        NSString *nimi=@"Malle";
        return nimi;
}
//
-(int)meetodCTyypiMuutujagaJaParameetriga:(int)t2isarv{
        int vastus=2*t2isarv;
        return vastus;
}

-(NSInteger)meetodObjcTyypiMuutujagaJaKaheParameetriga:(NSInteger)arv1
parameeter2:(NSInteger)arv2{

    NSInteger vastus=arv2+arv1;
        return vastus;
}
```

**Koodinäide 9. Erinevat tüüpi meetodid.**

## 3.3 Objektide lähtestamine

Objective-C keeles algavad objekte lähtestavad meetodid (tuntumates programmeerimiskeeltes tuntud kui konstruktorid) eesliitega **init**.

Objekti lähtestamisel kutsutakse välja meetodiga **[super init]** super klass. Super klassi väljakutsumine tagab selle, et eksemplaride muutujad, mis on määratud kõrgemate klasside poolt pärimisahelas initsialiseeritakse esimesena. Konstruktori loomiseks defineerime **MinuObjekt** päisefailis konstruktori(vt Koodinäide 10).

```
-(id)initWithNimi:(NSString *)uusNimi Perekonnanimi:(NSString *)uusPerenimi;
```

**Koodinäide 10. Konstruktor meetodi defineerimine päisefailis**

Põhifailis implementeerime päisefailis defineeritud konstruktori, kuhu on võimalik objekti loomisel parameetritena kaasa anda nimi ja perekonnanimi. Loome ka meetodi, mis kasutab konstruktoris väärtuse saanud muutujaid nimega **t2isNimi** ning kutsume funktsiooni **t2isNimi** oma loodud konstruktoris välja (vt Koodinäide 11).

```
-(id)initWithNimi:(NSString *)uusNimi Perekonnanimi:(NSString *)uusPerenimi{

   self=[super init];
   if(self){

      nimi=uusNimi;
      perekonnanimi=uusPerenimi;
         NSString *eesJaPerekonnaNimi=[self t2isNimi];
      NSLog(@"%@",eesJaPerekonnaNimi);
   } //if lause lõpp


   return self;
}//Konstruktor meetodi lõpp

-(NSString *)t2isNimi{

   //Liidame kaks string kokku.
   return [NSString stringWithFormat:@"%@ %@",nimi,perekonnanimi];
}
```

**Koodinäide 11. Oma loodud objekti lähtestamise meetod**

Loodud meetodis liidame me ka **stringWithFormat** meetodi abil nimi ja perekonnanimi sõned üheks sõneks.

```
MinuObjekt *minuObjekt=[[MinuObjekt alloc]initWithNimi:@"Kalle" Perekonnanimi:@"Malle"];
```

**Koodinäide 12. Loodud objekti lähtestamine**

Muudame objekti initsiaatori **ObjectiveCAppDelegate** põhifailis enda loodud initsiaatoriks (Koodinäide 12. Loodud objekti lähtestamine).

## 3.4 Massiivid,tsüklid ja sõnastikud

Selles peatükis õpime kasutama Objective-C massiive, tsükleid ja sõnastikke. Kõigepealt tutvustame lühidalt *Key-Value Coding* mehhanismi. Paljud objektid Objective-C keeles kasutavad *Key-Value Coding* mehhanismi, et selle abil kaudselt objektide atribuutidele ligi pääseda. Näiteks kasutab seda **NSDictionary** tüüpi objekt, kus on võimalik võtme abil atribuutidele ligi pääseda.

*Key-value coding* ehk lühidalt KVO on võtme ja väärtuse abil koodimise mehhanism, mille abil pääseb objektide atribuutidele ligi kaudselt, kasutades selleks sõnesid, et atribuute ära tunda (Apple Inc., 2012).

Massiivi loomiseks avame eelmises alampeatükis loodud projekti. Loome uue muudetava massiivi, nimega **NimedeMassiv**. Järgmiseks loome veel kolm **MinuObjekti** ja lisame need massiivi (vt Koodinäide 13).

```
MinuObjekt *minuObjekt2=[[MinuObjekt alloc]init];
 minuObjekt2.nimi=@"Kalle";
 minuObjekt2.perekonnanimi=@"Kask";
 MinuObjekt *minuObjekt3=[[MinuObjekt alloc]init];
 minuObjekt3..nimi=@"Malle";
 minuObjekt3.perekonnanimi=@"Kask";

 NSMutableArray *nimedeMassiiv=[[NSMutableArray alloc]init];
 [nimedeMassiiv addObject:minuObjekt];
 [nimedeMassiiv addObject:minuObjekt2];
 [nimedeMassiiv addObject:minuObjekt3];
```

**Koodinäide 13. Objektide lisamine massiivi**

Nüüd väljastame vastavad objektid *foreach* tsükli abil välja ja *for* tsükliga eemaldame massiivist objekti, mille atribuudile **nimi** on antud väärtuseks Kalle. Massiivi suuruse saab teada meetodiga **count** ja sõnede võrdlemiseks meetodit **isEqualToString**. Kui objekt sisaldab kahte muutujat, nime ja perekonnanime. Selle, millist objekti vastaval indeksil soovime, saame *key-value coding* abil öelda meetodiga **valueForKey** (vt koodinäide 9) .

```
for(int i=0;i<nimedeMassiiv.count;i++){

    if([[[nimedeMassiiv objectAtIndex:i] valueForKey:@"nimi"] isEqualToString:@"Kalle"]){

       [nimedeMassiiv removeObjectAtIndex:i];

    }
```

```
    }

  for(MinuObjekt *objekt in nimedeMassiiv){

     NSLog(@"Nimed %@",objekt.nimi);

  }
```

**Koodinäide 14. Objektide eemaldamine massiivist**

Kui me oleksime soovinud luua **NSArray** tüüpi massiivi objekti, siis sinna oleks tulnud massiivi lähtestamisel objektid sisestada (Koodinäide **15**. NSArray tüüpi massiiv).

```
NSArray  *tavalineMassiiv=[[NSArray alloc]
initWithObjects:@"Objekt1",@"Objekt2",@"Objekt3",@"jne",nil];
```

**Koodinäide 15. NSArray tüüpi massiivs**

Järgnevalt loome **NSDictionary** tüüpi massiivi, kus andmeid salvestatakse võtme väärtustega. Näiteks väärtuseks on inimese nimi sõne tüüpi objekt ja võtmeks nimi. Samuti loome **NSMutableDictionary** tüüpi massiivi, kus on võimalik andmeid muuta (vt Koodinäide 16. NSDictionatry ja NSMutableDictionary sõnastik).

```
NSDictionary *sonastik=[[NSDictionary alloc]
initWithObjectsAndKeys:@"väärtus",@"Võti",@"kalle",@"nimi",nil];
 NSMutableDictionary *muudetavSonastik=[[NSMutableDictionary alloc] init];
 [muudetavSonastik setObject:@"kalle" forKey:@"nimi"];
```

**Koodinäide 16. NSDictionatry ja NSMutableDictionary sõnastik**

## Ülesanded

1. Looge uus klass nimega inimene, muutujatega vanus ning eesnimi. Looge kaks meetodit, millest üks lisab inimese vanusele 10 aastat ning teine meetodi millele on võimalik parameetrina kaasa anda inimese perekonnanimi. Mõlema meetodi poolt väljastatud väärtused printige konsooli aknasse.
2. Lisage eelmises ülesandes loodud klass eelmises peatükis loodud projektile. Lisamiseks valige oma projekti kaustast vastav klass ning lohistage hiirega eelmises peatükis loodud klassi kausta. Lisage perekonnanime lisamise meetod nupu tegevust kuulava funktsiooni sisse, ning väljastage tulemus sildi (ingl.k *label*) sisse nupu vajutusel.
3. Looge klassi meetod, mille abil saab perekonnanime muuta.

# 4 Tabel ja rippmenüü loomine

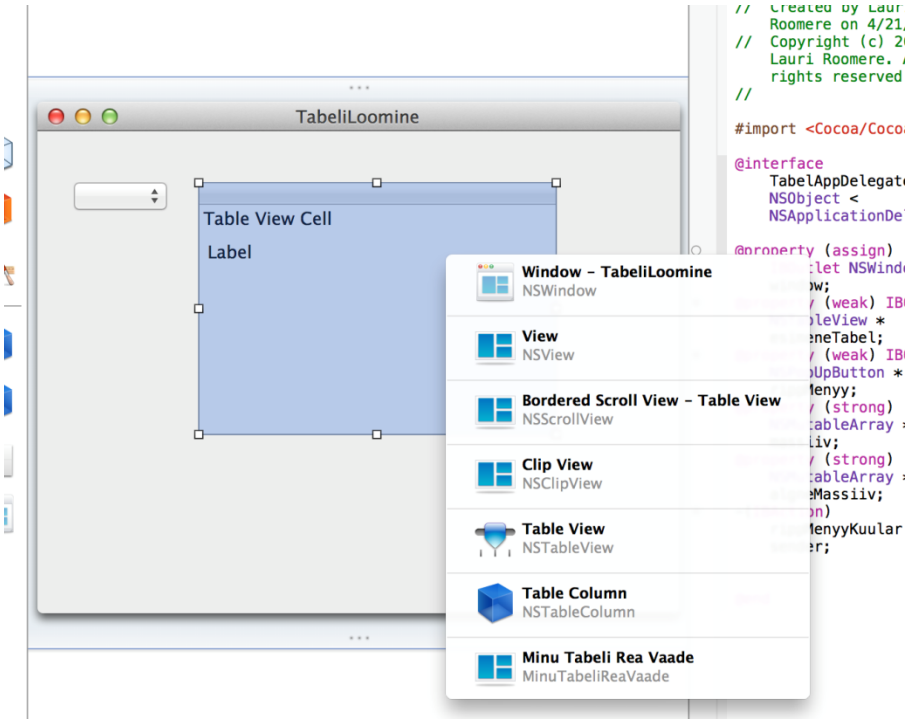
Selles peatükis õpime kasutama tabeli (**NSTable**) ja rippmenüü (**NSPopUpButton**) lihtsamaid funktsionaalsusi ja kinnistame eelmises peatükis õpitut. Samuti näitame kuidas ja miks kasutame delegaate ja andmeallikaid (ingl.k *data sources).*

Peatükis kasutame vastavaid elemente seetõttu, et need on kasutajaliidese tavapärasemad elemendid ning selle peatüki lõpuks on õppuril oskus luua lihtsamat tüüpi tabelit, rippmenüüd, tekstivälja, silti ja nuppu ning samuti teab õppur kasutajaliidese elementidega seotud põhilisemaid meetodeid. Vastavate kasutajateliideste abil suudavad õppurid kuvada andmeid kasutajale ning teatud kasutajaliidese elementidega toimuvaid sündmusi kuulata.

## 4.1 Tabeli loomine

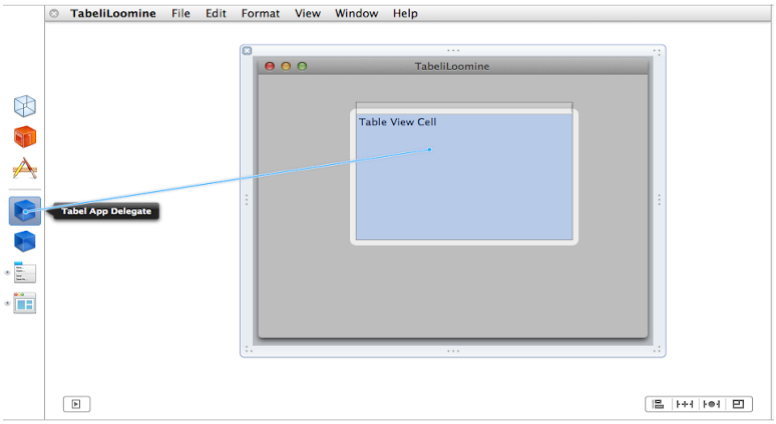
Kõige pealt loome uue projekti, millele anname nimeks meie näidises näiteks Tabel ja eesliite nimeks samuti Tabel. Seejärel avame **xib** faili ning pukseerima kasutajaliidese objektide kogust aknale **NSTableView** objekti.

Järgmiseks klikime oma tabeli objekti peale ctr+shift klahvi kombinatsiooni all hoides ning parema nupuga klikkides ilmub uus aken. Uues aknas on näidatud kõik kasutajaliidese elemendid hierarhiliselt, millest tabel koosneb või mille sees see on, valige valikust **Table View**, et me saaksime luua **Outlet**'i oma tabelile (vt Joonis 18).



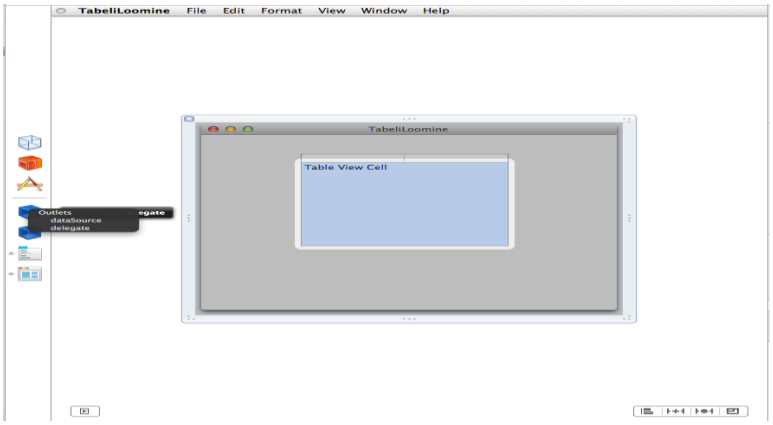

Joonis 18. Kasutjaliidese elementide vaade, mis on seotud tabeliga.

Seejärel ühendame tabeli *delegate* ja *data source* meetodi aknal oleva klassi objektiga millel on nimeks **Tabel App Delegate** (vt Joonis 19).



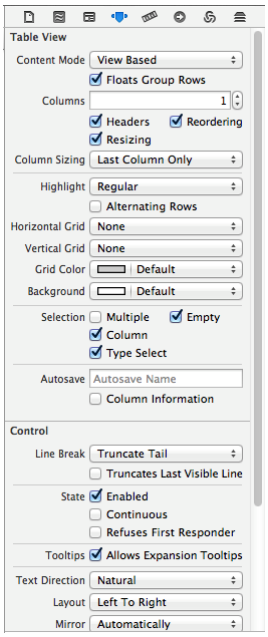

**Joonis 19. Tabeli ühendamine Tabel App Delegate objektiga.**

Delegaadi abil ütleme, et **Tabel App Delegate** objekt kuulab tabeli objekti sündmusi ning sündmuste korral teavitab **Tabel App Delegate** objekti sellest. *Data source* näitab kust klassist tulevad andmed vastavasse tabelisse (vt
Joonis 20).




**Joonis 20. Tabeli ühendamine *dataSource* ja delegaadiga.**

Valime utiliitide paneelist *File's Inspector'i*, kus me saame kirjeldada kasutajaliidese elemendi atribuutide omadusi (vt Joonis 21).




**Joonis 21. Tabeli omaduste kirjeldamine**

Objekti peale vajutades nägite kuidas *File's Inspectori's* muutus pidevalt objekti klassi nimetus. *Content Mode*'iks valime view based ja veergude (ingl.k *columns*) arvuks valime 1. Tähtis on alati jälgida, et õiged objektid oleksid omavahel sidemetega, vastasel korral rakendus lihtsalt ei käivitu või ei näe te soovitud tulemusi. Kõigepealt loome uue atribuudi nimega **nimedeMassiiv** oma päisfaili.

```
#import <Cocoa/Cocoa.h>
@interface TabelAppDelegate : NSObject <NSApplicationDelegate>
@property (assign) IBOutlet NSWindow *window;
@property (weak) IBOutlet NSTableView *esimeneTabel;
@property (strong) NSMutableArray *nimedeMassiiv;
@end
```

**Koodinäide 17. nimedeMassiivi lisamine**

Põhifailis loome meetodi nimega **lisameObjektidMasiiviJaLaemaTabelisse** , kus tekitame kaks **NSDictionary** objekti, millesse lisame inimese perekonnanime ja nime. Seejärel lisame loodud **NSDictionary** objektid massiivi ja laeme andmed tabelisse. (vt Koodinäide 18).

```
// TabelAppDelegate põhifaili kood


- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    [self lisameObjektidMasiiviJaLaemeTabelisse];
}

-(void)lisameObjektidMasiiviJaLaemeTabelisse{


    nimedeMassiiv =[[NSMutableArray alloc]init];

    NSDictionary *sonastik=[[NSDictionary alloc]initWithObjectsAndKeys:@"Mati",@"Nimi", nil];
    NSDictionary *sonastikKaks=[[NSDictionary alloc]initWithObjectsAndKeys:@"Malle",@"Nimi",
nil];

    [nimedeMassiiv addObject:sonastik];
    [nimedeMassiiv addObject:sonastikKaks];
    [self.esimeneTabel reloadData];

}
```

**Koodinäide 18. Objektide massiivi lisamine ja tabeli laadimine**

Kui massiiv on valmis loodud alustame tabeli osa koodiga. Kõigepealt defineerime ära **numberOfRowsInTableView** meetodis palju ridasid me oma tabelis soovime näidata. **numberOfRowsInTableView** meetod tagastab tabeli ridade arvu (vt Koodinäide 19. Tabeli näidatavate ridade määramine.).

```
-(int)numberOfRowsInTableView:(NSTableView *)tableView{

    NSNumber *massiiviSuurus=[NSNumber numberWithInteger: nimedeMassiiv.count];

    return [massiiviSuurus intValue];

}
```

**Koodinäide 19. Tabeli näidatavate ridade määramine.**

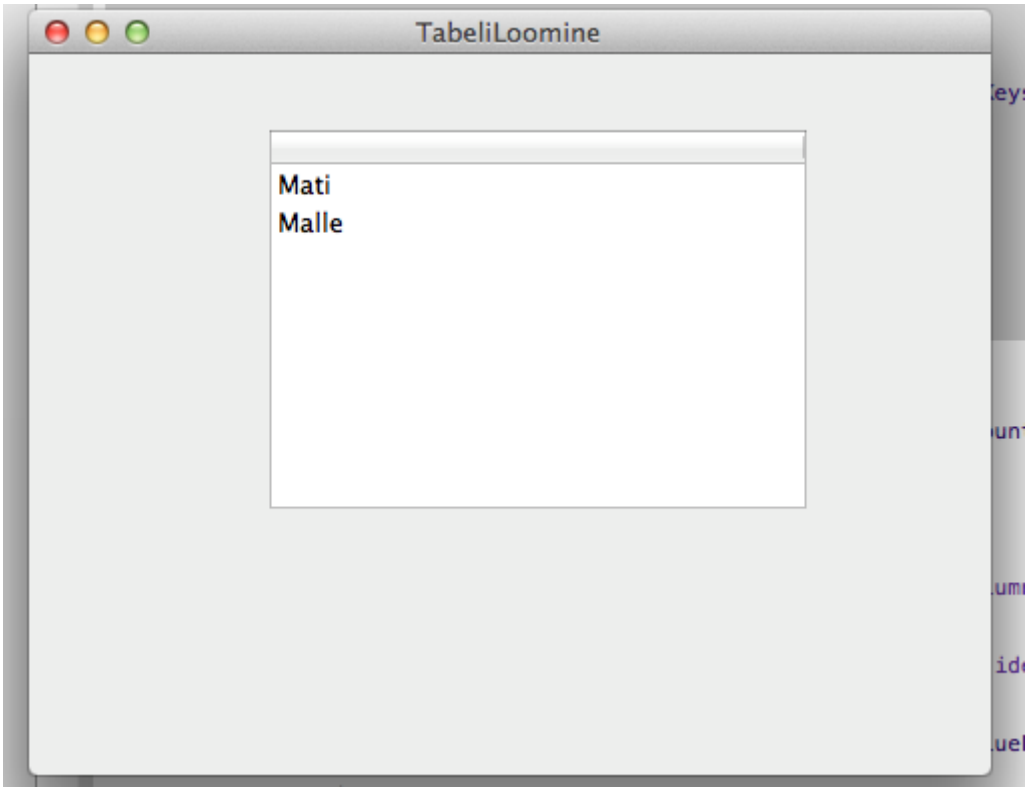
Järgmise meetodiga **-(id)tableView: (NSTableView*) table viewForTableColumn: (NSTableColumn *) column row:(int)row** kirjeldame tabelile mida soovime tabeli vaates olevatel ridadel näidata (vt Koodinäide 20).

```
-(id) tableView:(NSTableView *)table viewForTableColumn:(NSTableColumn *)column row:(int)row{
    NSTableCellView *reaVaade=[table makeViewWithIdentifier:column.identifier owner:self];

    reaVaade.textField.stringValue=[[ nimedeMassiiv objectAtIndex:row] valueForKey:@"Nimi"];
```

```
    return reaVaade;

}
```

**Koodinäide 20. Tabeli real olevatele objektidele väärtuste määramine.**

Lõpptulemusena kuvatakse meie tabelis kasutajale nimesid Mati ja Malle (vt Joonis 22).



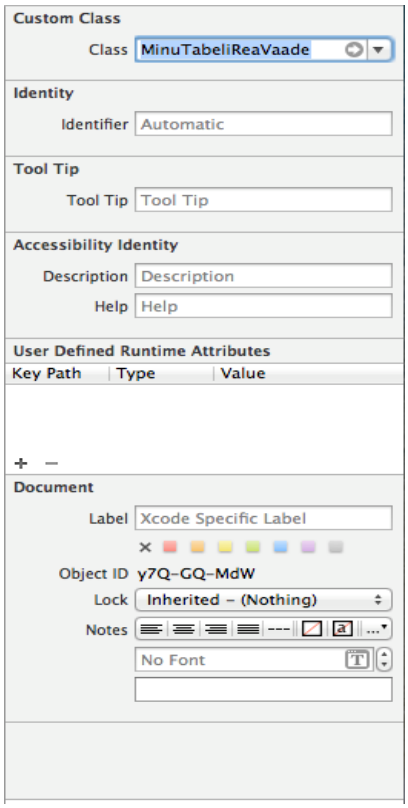

**Joonis 22. Tabeli loomise lõpptulemus**

Kui me soovime näidata rohkem erinevat tüüpi objekte oma tabeli rea vaates, siis selleks peame looma uue klassi, mis pärib **NSTableCellView** objekti omadused. Seejärel kirjeldame päisefailis kasutajaliidese elemendi atribuudi mida soovime vastavasse tabeli ritta lisada (vt Koodinäide 21). Loome uue klassi, millele anname nimeks **MinuTabeliReaVaade** ning lisame uue **NSTextField'**i tüüpi **outlet**'i atribuudi **MinuTabeliReaVaade.h** faili. Samuti nimetame **NSObject**'i ümber **NSTableCellView**'ks, kuna me soovime kasutada vastavat objekti oma loodud tabeli ridade vaate näitamiseks.

```
@interface MinuTabeliReaVaade : NSTableCellView
@property (strong) IBOutlet NSTextField *teineTekstiVali;
@end
```

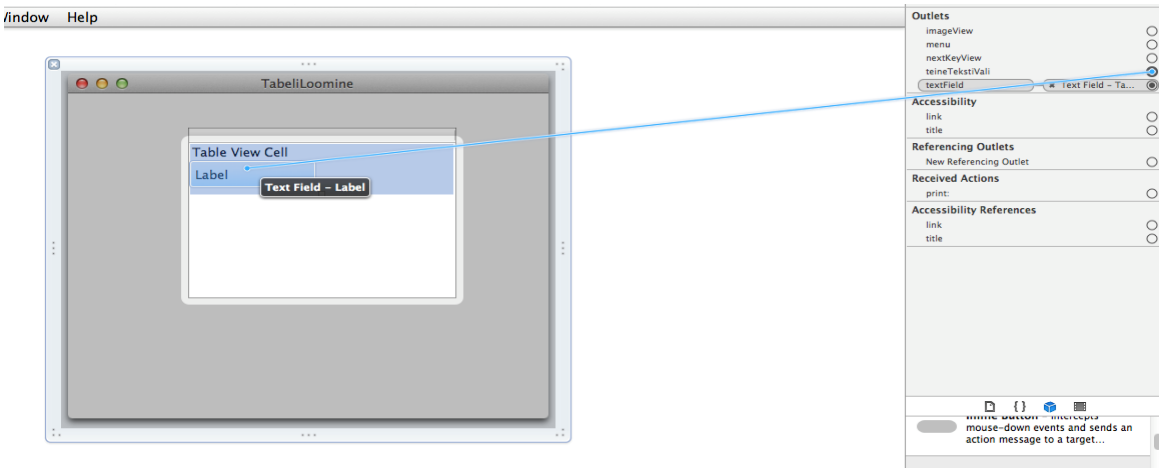
**Koodinäide 21. MinuTabeliReaVaade.h failis uue objekti atribuudi määramine**

Pärast seda avame oma **xib** faili, klikime neli korda tabeli peale ja veendume et valisime **NSTableCellView** objekti. Avame paremal üleval paneelist *Custom Class* (tavaline klass) vaate ja lisame sinna enda loodud klassi nimetuse **MinuTabeliReaVaade** (vt Joonis 23).




Joonis 23. Klassi omaduste määramine

Seejärel lisame tabeli reale juurde ühe **NSTextField** objekti ja ühendame selle **NSTableViewCell**'iga, avades paremal olevast paneelist *Connections Inspectori* ja ühendage hiirega lohistades see **teineTekstiVali** objekti külge (vt Joonis 24. Kasutajaliidese objekti (teineTekstiVali) ühendamine MinuTabeliReaVaade objektiga.).



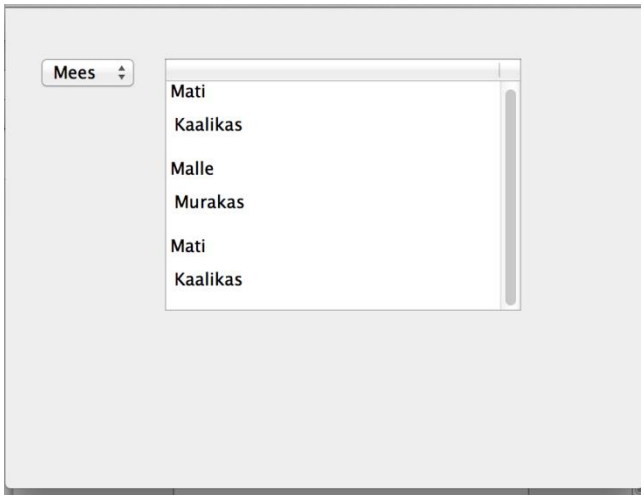

Joonis 24. Kasutajaliidese objekti (teineTekstiVali) ühendamine MinuTabeliReaVaade objektiga.

Kui objekt on ühendatud, impordime oma loodud klassi **MinuTabeliReaVaade.h** **TabelAppDelegate** klassi ja lisame juurde koodiosa, mis annab väärtuse ka meie poolt loodud tekstiväljale ning muudame **NSTableCellView** tekitamise meetodist saadava objekti **MinuTabeliReaVaade** objektiks (vt Koodinäide 22).

```
-(id) tableView:(NSTableView *)table viewForTableColumn:(NSTableColumn *)column
row:(int)row{

   MinuTabeliReaVaade *reaVaade=[table makeViewWithIdentifier:column.identifier
owner:self];

   reaVaade.textField.stringValue=[[massiiv objectAtIndex:row] valueForKey:@"Nimi"];
   reaVaade.teineTekstiVali.stringValue=[[massiiv objectAtIndex:row]
valueForKey:@"Perenimi"];
   return reaVaade;

}
```

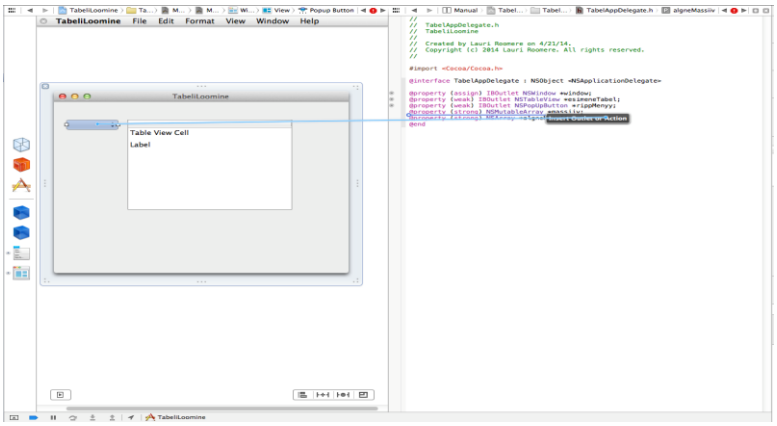
**Koodinäide 22. MinuTabeliReaVaade objekti kasutamine rea vaate kuvamiseks**

## 4.2 Rippmenüü loomine



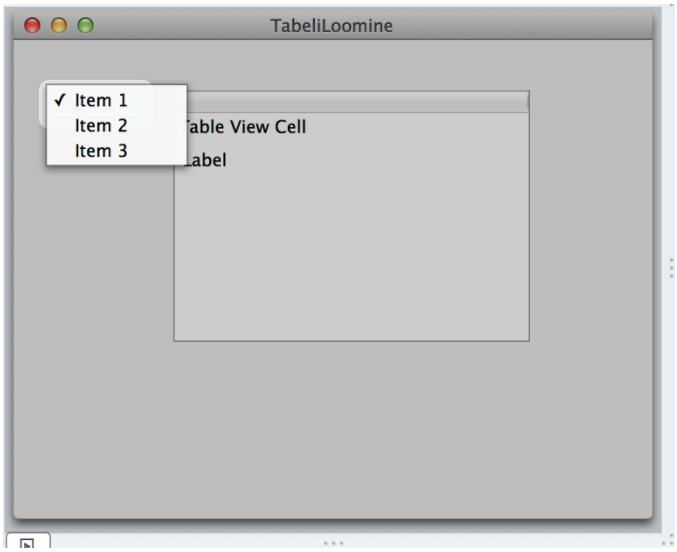

Rippmenüüd on hea kasutada selleks, et anda kasutajale ette kindel valik mingite kindlate sündmuste või olukordadega. Näiteks tabelis olevate andmete filtreerimiseks mingite kindlate kirjete abil või kasutaja staatuse näitamiseks (kas kasutaja on rakenduses sees, väljas jne).

Avage eelmises peatükis loodud projekt ja tirige hiirega objektide teegist aknale **NSPopUpItem**. Seejärel ühendage mõlema outlet'id oma **Table App Delegate** objektiga (vt Joonis 25).



Joonis 25. NSPopUpItem'i outleti tekitamine TabelAppDelegate.h päisefaili.

Seejärel vajutage loodud rippmenüü objekti peale kuni avaneb rippmenüü objektide aken , mis on ka kujutatud ning kustutage kõik objektid backspace'i või delete'i vajutades.




Joonis 26. Rippmenüüs vaikimis olevate objektide kustutamine.

Seejärel loome meetodi nimega **lisameObjektidRippMenyysse,** kus tekitame kaks uut objekti pealkirjadega „Naine" ja „Mees", mida vastavalt valides lisatakse tabelisse uus objekt (vt Koodinäide 23).

```
-(void)lisameObjektidRippMenyysse{

    [rippMenyy addItemWithTitle:@"Naine"];
    [rippMenyy addItemWithTitle:@"Mees"];

}
```

Koodinäide 23. Objektide lisamine NSPopUpItem'isse.

Seejärel kirjeldame ära tegevuse meetodi, mis kuulab millise pealkirjaga objekt on valitud ning vastavalt valitud objektile lisame tabelisse meesoost eesnimi või naissoost eesnimega isiku.

Nagu olete märganud algavad kõik tegevuste ja sündmustega seotud meetodi **IBAction** objektiga (vt Koodinäide 24. NSPopUpItem'I sündmuseid kuulav meetod).

```
-(IBAction)rippMenyyKuular:(id)sender{

   if([[[sender selectedItem] title] isEqualToString:@"Naine"]){

       NSDictionary                                                *sonastikKaks=[[NSDictionary
alloc]initWithObjectsAndKeys:@"Malle",@"Nimi",@"Murakas",@"Perenimi", nil];
      //Lisab uue objekti masiivi
        [massiiv addObject:sonastikKaks];

   }else{
      NSDictionary                                                *sonastik=[[NSDictionary
alloc]initWithObjectsAndKeys:@"Mati",@"Nimi",@"Kaalikas",@"Perenimi", nil];
      //Lisab uue objekti masiivi
       [massiiv addObject:sonastik];
   }
   //Laeb andmed tabelisse
   [self.esimeneTabel reloadData];
}// rippMenyyKuular meetodi lõpp
```

**Koodinäide 24. NSPopUpItem'I sündmuseid kuulav meetod**

Samuti tuleb lisada meie kirjeldatud sündmusi kuulav meetod päisefaili ning luua ühendus loodud meetodi ja rippmenüü objekti vahel (vt Koodinäide 25).

```
@interface TabelAppDelegate : NSObject <NSApplicationDelegate>

@property (assign) IBOutlet NSWindow *window;
@property (weak) IBOutlet NSTableView *esimeneTabel;
@property (weak) IBOutlet NSPopUpButton *rippMenyy;
@property (strong) NSMutableArray *massiiv;
-(IBAction)rippMenyyKuular:(id)sender
```

**Koodinäide 25. rippMenyyKuular lisamine päisefaili.**

## Ülesanded

1. Looge rakendus, kus on võimalik sisestada kaks arvu ning nupule vajutades sooritatakse korrutustehe. Tulemus väljastatakse tabelisse.
2. Lisage eelmises ülesandes loodud rakendusele juurde võimalus valida, millist tehet ( korrutamine, jagamine, liitmine, lahutamine) sooritatakse.

# 5 Andmete salvestamine

Selles peatükis õpime linkima oma projektiga välist teeki (FMDB) ning andmeid andmebaasi lisama. Lühidalt tutvustatakse ka *SQLite Manager'i* mille abil on lihtne andmebaasi hallata. Saame veenduda, et loodud andmebaasi on ka uusi kirjeid lisatud ja need on õigesti salvestatud tabelisse.

## 5.1 FMDB linkimine projektiga

Kõigepealt laadige alla FMDB kokkupakitud fail aadressilt https://github.com/ccgus/fmdb. Pakkige zip fail lahti kohta, kus selle kergesti üles leiate. Seejärel avage *fmdb-master->src* kaust ning kopeeriga seal oleva fmdb kaust oma tekitatud projekti kausta. Kui olete eelneva toimingud lõpetanud, lohistage projekti kausta kopeeritud fmdb kaust projekti navigaatorisse. Nüüd on võimalik meil kasutada FMDB teeki oma rakenduses, kuid FMDB teek kasutab *libsqlite3.0.dylib* teeki, mille abil on võimalik SQL keelt rakenduses kasutada.

Pärast FMDB teegi lisamist projekti navigaatorisse avaneb uus aken, kus on võimalik valida lisatud failidele seadistus. Kui seadistus on paigas vajutage Finish.

Seejärel impordime oma **SalvestamineAppDelegate.h** päisefailis **FMDatabase.h** klassi, et me saaksime FMDB teeki oma rakenduses kasutada (vt Koodinäide 26).

```
#import <Cocoa/Cocoa.h>
#import "FMDatabase.h"
@interface SalvestamineAppDelegate : NSObject <NSApplicationDelegate>
```

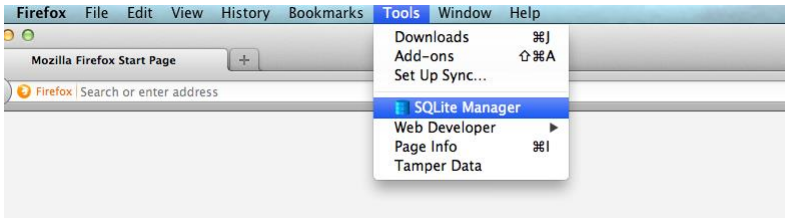
**Koodinäide 26. FMDB importimine**

Järgmisena laadige alla mozilla firefoxi *SQLite Manager*, mille abil me saame oma loodud andmebaasi vaadata ja modifitseerida. *SQLite Manager* on allalaetav aadressilt https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/.

*SQLite Manager*'i kasutame seetõttu, et seda on lihtne kasutada ja see sisaldab ülesande sooritamiseks vajalikku funktsionaalsust ning see on tasuta tarkvara.

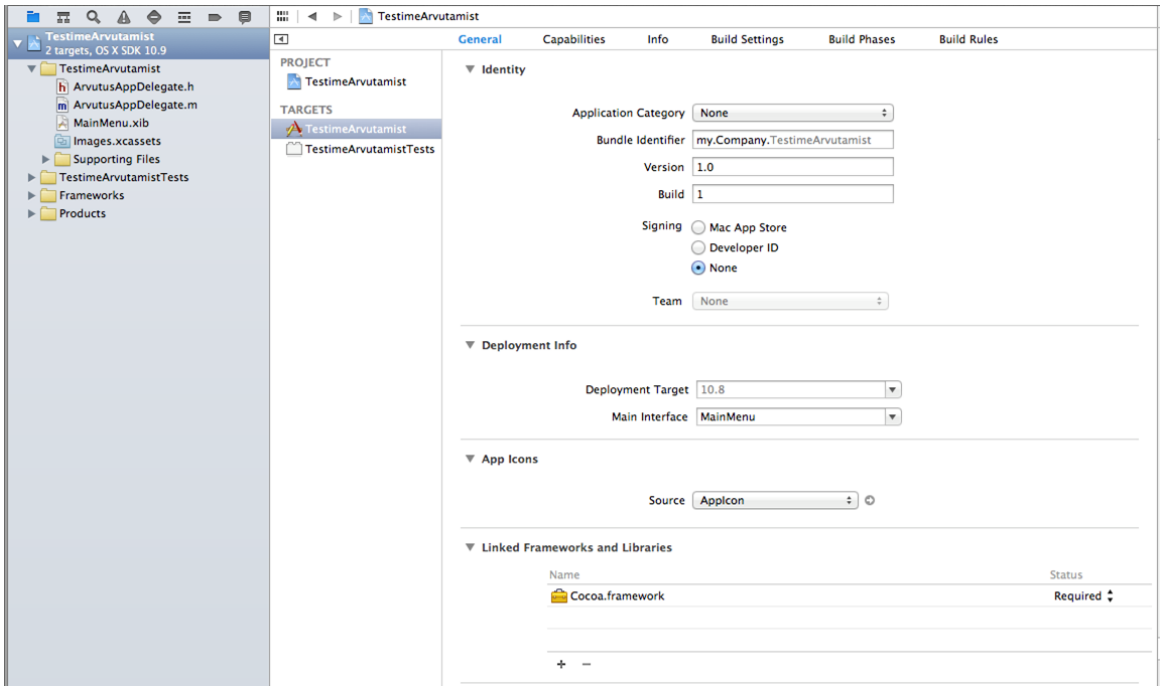
Avage *Mozilla Firefox*'i brauser ning valige ülevalt menüüst *tools->SQLite Manager*. Loome uue faili valides ülevalt programmi menüüst *New Database* ja avanenud aknas anname andmebaasile nimetuse. Nimetusele liidetakse automaatselt otsa .sqlite faililaiend, seda ei ole ise vaja lisada (vt Joonis 27). Salvestage loodud andmebaas oma projekti kausta ja lohistage see

sealt projekti navigaatorisse. Andmebaasi projekti navigaatorisse vedamisega tagame selle, et rakendust käitades lisatakse see rakenduse kausta.



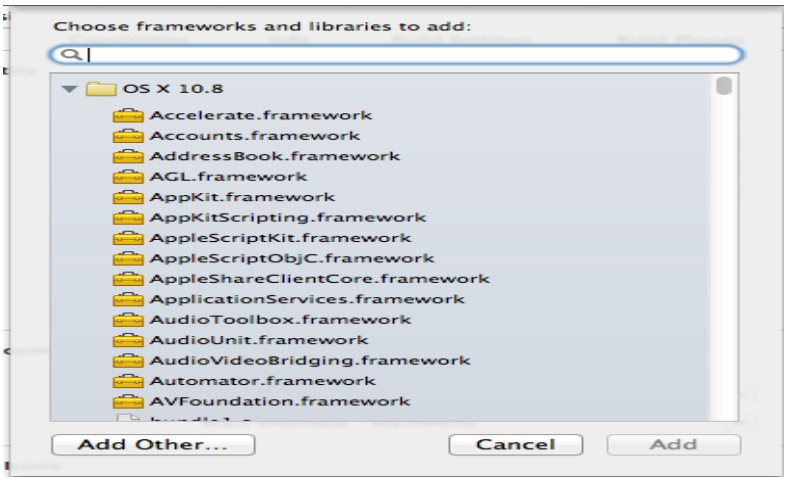

Joonis 27. SQLite Manageri avamine

Seejärel valige *Execute SQL* ning kirjutage vastavasse välja uue tabeli loomise päring CREATE TABLE "EsimeneTabel" ("Perenimi" VARCHAR(128) DEFAULT (null) ,"Nimi" VARCHAR(128) DEFAULT (null) ,"Vanus" INTEGER DEFAULT (0) ) ja vajutage (mida). SQL käsuga tekkis meile uus tabel, kuhu saame hakata lisama inimeste andmeid.




Joonis 28. Teegi lisamine

Otsitav teek on kätte saadav Xcode arenduskeskkonnas olemas olevate teekide seast. Pärast pluss märgiga nupu  vajutamist, mis on teegi lisamisenupp, kirjutage avanenud akna otsinguribale libsqlit, et kitsendada valiku tulemusi ning valige *libsqlite3.0.dylib* teek ja vajutage add (vt Joonis 29).




Joonis 29. Teegi lisamine 2

## 5.2 Rakenduse ühendamine andmebaasiga

Ühenduse loomiseks peame teadma teed meie loodud andmebaasini (andmebaasi paigutasime me oma projekti kausta). Tee loodud andmebaasi on kergesti kirjeldatav **NSBundle** objekti abil. **NSBundle** objekti abil saame kindlaks määrata, kus meie andmebaas projekt asub *Resource* kaustas, mis tekitatakse käitamise ajal ning mis sisaldab kõiki projektiga sisalduvaid faile (vt Koodinäide 28).

```
NSString *teeAndmebaasi=[[NSBundle mainBundle] pathForResource:@"Andmebaas1" ofType:@"sqlite"];
```

Koodinäide 27. Andmebaasi faili tee leidmine

Kui meile on teada, kus meie andmebaas asub, võime alustada andmebaasiga ühenduse loomist. Selle jaoks tuleb meil tekitada **FMDatabase** objekt ning anda talle tee meie andmebaasini. Ühenduse loomiseks tekitame uue meetodi mis tagastab loogikamuutuja väärtuse ning tekitame päisefaili atribuudi nimega andmebaas, mille sünteesime(**@synthesize**) oma põhifailis.

Objective-C on loogikamuutuja väärtusteks traditsioonilise TRUE/FALSE asemel YES/NO.

```
// SalvestamineAppDelegate.h
#import <Cocoa/Cocoa.h>
#import "FMDatabase.h"
@interface SalvestamineAppDelegate : NSObject <NSApplicationDelegate>

@property (assign) IBOutlet NSWindow *window;
@property FMDatabase *andmebaas;
@end
```

```
// SalvestamineAppDelegate.m
```

```
-(BOOL)loomeYhenduseAndmebaasiga{

NSString *teeAndmebaasi=[[NSBundle mainBundle] pathForResource:@"Andmebaas1" ofType:@"sqlite"];
   andmebaas= [FMDatabase databaseWithPath:teeAndmebaasi];

   if([andmebaas open]){


      return YES;

   }else{

      return NO;
   }

}// loomeYhenduseAndmebaasiga meetodi lõpp
```

**Koodinäide 28. Ühenduse loomine andmeaasiga**

Meetod **[andmebaas open]** avab andmebaasi ning tagastab samuti **YES/NO** väärtuse vastavalt sellele, kas andmebaas avati või esines avamisel mõni viga.

Kutsume **loomeYhenduseAndmebaasiga** meetodi **applicationDidFinishLaunching** meetodis välja kontrollides kas ühendus loodi või mitte (vt Koodinäide 28).

Loome uue meetodi, mille abil lisada andmebaasi andmeid. Meetodile anname nimeks näiteks **lisaAndmedTabelisseEsimeneAlustadesNimi** ja kirjeldame kolm parameetrit (nimi, perekonnanimi, vanus).

```
-(void)lisaAndmedTabelisseEsimeneAlustadesNimi:(NSString *)nimi Perenimi:(NSString *)perenimi Vanus:(int)vanus{

   BOOL result=[andmebaas executeUpdate:@"INSERT INTO EsimeneTabel(Perenimi,Nimi,Vanus)VALUES(?,?,?)",nimi,perenimi,[NSNumber numberWithInt:vanus ]];

   if(!result){NSLog(@"Andmeid andmebaasi sisestades esines tõrge");}else{NSLog(@"Andmed salvestati baasi");}
}
```

**Koodinäide 29. Andmete lisamine andmebaasi**

Loodud meetodis salvestatakse andmed meie loodud andmebaasi tabelisse meetodiga **executeUpdate**. **ExcecuteUpdate** meetod tagastab ka loogikamuutuja väärtuse selle kohta, kas andmete salvestamine õnnestus või mitte.

Seda kas andmete salvestamisel esines vigu kontrollime *if* lausega. Result muutuja ees olev märge **!** tähendab, et juhult kui loogikamuutuja väärtuseks on **NO,** siis prinditakse välja sõnum *„Andmeid andmebaasi sisestades esines tõrge"* (vt Koodinäide 29).

Kui andmebaasi salvestamise jaoks on meetod loodud, siis loome uue meetodi, mille abil saame andmeid andmebaasist küsida. Meetodile anname nimeks **kysimeAndmedAndmebaasistAndmed**.

Andmete küsimiseks kasutame meetodit **executeQuery**, mille abil saame sooritada andmebaasist erinevaid päringuid. Andmebaasist saadud andmed tagastatakse **NSSet** objektina. Vastava objekti käime läbi *while* tsükli abil ning saadud andmed lisame loodud massiivi nimega massiiv. Meetodi **stringForColumn** abil saame öelda, millise veeru nimega seotud muutuja väärtust soovime oma andmebaasist. Kuna vanus tagastatakse andmebaasist täisarvuna, tuleb see ümber konverteerida **NSNumber** tüüpi objektiks käsuga **numberWithInt**.

Kui andmed on andmebaasist kätte saadud, tagastab meetod **NSMutableArray** tüüpi massiivi (vt Koodinäide 30).

```
-(NSMutableArray *)kysimeAndmedAndmebaasistAndmed{

    FMResultSet *tulemus= [andmebaas executeQuery:@"SELECT * FROM EsimeneTabel"];
    NSMutableArray *massiiv=[[NSMutableArray alloc]init];
    while ([tulemus next]) {

        NSString *nimi=[tulemus stringForColumn:@"Nimi"];
        NSString *perenimi=[tulemus stringForColumn:@"Perenimi"];
        NSNumber *vanus=[NSNumber numberWithInt:[tulemus intForColumn:@"Vanus"]];
        NSDictionary *uusObjekt=[[NSDictionary
alloc]initWithObjectsAndKeys:nimi,@"Nimi",perenimi,@"Perenimi",vanus,@"Vanus",nil];
        [massiiv addObject:uusObjekt];
    }

    return massiiv;
}
```

**Koodinäide 30. Andmete küsimine andmebaasist**

Nüüd on vaja vastavad meetodid **applicationDidFinishLaunching** meetodis välja kutsuda ning andmebaasist saadud tulemused konsooli väljastada. Saadud andmed saame välja printida for tsüklit ja **objectAtIndex** meetodit kasutades, mis tagastab massiivist vastaval indeksil oleva objekti (vt Koodinäide 31).

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    if([self loomeYhenduseAndmebaasiga]){
```

```
    [self lisaAndmedTabelisseEsimeneAlustadesNimi:@"Mati" Perenimi:@"Kask" Vanus:18];
    NSMutableArray *uusMassiiv=[self kysimeAndmedAndmebaasistAndmed];

    for(int i=0;i<uusMassiiv.count;i++){

        NSLog(@"Nimi %@",[[uusMassiiv objectAtIndex:i] valueForKey:@"Nimi"]);
        NSLog(@"Perekonnanimi %@",[[uusMassiiv objectAtIndex:i] valueForKey:@"Perenimi"]);
        NSLog(@"Vanus %@",[[uusMassiiv objectAtIndex:i] valueForKey:@"Vanus"]);
    }

  }else{

    NSLog(@"Ei suudetud luua ühendust andmebaasiga");

  }
}
```

**Koodinäide 31. Andmete väljastamine**

## Ülesanded

1.Looge kasutajaliides andmete lisamiseks ja näitamiseks. Andmete näitamiseks kasutage tabelit ja lisamiseks tekstivälja ja nuppu. Kui olete tuttavamaks saanud iseseisvalt mõne teist tüüpi kasutajaliidesega, võite vabalt ka neid kasutada andmete lisamiseks.

2. Looge kaks uut meetodit millest üks on andmete kustutamiseks ja teine uuendamiseks. Värskendage ka vastavate toimingute abil tabelit. Kustutamiseks võiks nupu lisada tabelisse. Seejärel on võimalik kasutajal valitud andmeid kergesti kustutada. Vastava objekti leidmiseks, kes saatis tegevusest sõnumi, tagastab rea numbri **[NSTableView selectedRow],** mille järgi on võimalik massiivist vastav objekt kustutada.

# 6 Komponenttestimine

Selles peatükis loome uue rakenduse millele lisame komponenttestid (ingl.k *unit testing*), mille abil hakkame loodud meetodeid testima. Uue rakenduse loome seetõttu, et peatüki lõpus olevates ülesannetes luuakse rakendusele juurde kasutajaliidese objektid ja loogika.

Loome uue projekti millele paneme nimeks TestimisNaide ning eesliiteks rakenduse delegaadile lisame Testimine. Loome uue klassi nimega Kolmnurk, kus on meetodid kolmnurga ümbermõõdu ja pindala arvutamiseks (vt Koodinäide 32).

```
@interface Kolmnurk : NSObject
@property  NSInteger  kylgA;
@property  NSInteger  kylgB;
@property  NSInteger  kylgC;
-(NSNumber *)kolmnurgaPindalaArvutamineKorguse:(NSInteger)korgus Alus:(NSInteger)alus;
-(NSInteger)kolmnurgaYmbermoot;
-(id)initKolmnurkKylgA:(NSInteger)uusKylgA                    KylgB:(NSInteger)uusKylgB
KylgC:(NSInteger)uusKylgC;

@implementation Kolmnurk
@synthesize kylgA;
@synthesize kylgB;
@synthesize kylgC;
-(id)initKolmnurkKylgA:(NSInteger)uusKylgA                    KylgB:(NSInteger)uusKylgB
KylgC:(NSInteger)uusKylgC{
   //Konstruktor
   self=[super init];
   if(self){

      kylgA=uusKylgA;
      kylgB=uusKylgB;
      kylgC=uusKylgC;


   }

   return self;
}
-(NSNumber *)kolmnurgaPindalaArvutamineKorguse:(NSInteger)korgus Alus:(NSInteger)alus{
```

```
    NSNumber *vastus;
    vastus=[NSNumber numberWithFloat:0.5*(alus*korgus)];

    return vastus;
}

-(NSInteger)kolmnurgaYmbermoot{

    NSInteger vastus=kylgA+kylgB+kylgC;

    return vastus;

}
```

**Koodinäide 32. Kolmnurga klassi põhifaili ja päisefailis oleva kood**

Nüüd lähtestame **applicationDidFinishLaunching** meetodis **Kolmnurk** objekti ja kutsume loodud meetodid välja (vt Koodinäide 32). Kõik Objective-C test meetodid algavad sõnaga test ning seejärel järgneb test meetodi nimetus. Test meetodid on alati **void** tüüpi, kuna nad ei tagasta väärtusi.

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    Kolmnurk *kolmnurk=[[Kolmnurk alloc]initKolmnurkKylgA:5 KylgB:4 KylgC:4];
    NSLog(@"Kolmnurga ümbermõõt %li",(long)[kolmnurk kolmnurgaYmbermoot]);

    NSLog(@"Kolmnurga pindala %@",[kolmnurk kolmnurgaPindalaArvutamineKorguse:2 Alus:3]);

}
```

**Koodinäide 33. Kolmnurk objekti lähtestamine ja kasutamine**

Seejärel navigeerime kausta nimega **TestimisNaideTests** ja kustutame faili nimega **TestimisNaideTest** ning loome uue testimis juhtumi klassi nimega **KolmnurgaTest** tirides faili eksemplaride teegist Objective-C *test case class*'i oma projekti navigaatorisse vastavasse kausta. Testides vaatame, kas meetodid arvutavad pindala ja ümbermõõtu õigesti. Selle jaoks kasutame XCTest raamistikus olevaid meetodeid, millega võrdleme, kas meetodid tagastavad vastava sisendi korral õiged väärtused.

```
#import <XCTest/XCTest.h>
#import "Kolmnurk.h"
```

```
@interface KolmnurgaTest : XCTestCase
@property Kolmnurk *kolmnurk;
@end

@implementation KolmnurgaTest
@synthesize kolmnurk;
- (void)setUp
{
    [super setUp];
    // Put setup code here. This method is called before the invocation of each test method in the class.
    kolmnurk=[[Kolmnurk alloc] initKolmnurkKylgA:5 KylgB:4 KylgC:4];
    XCTAssertNotNil(kolmnurk, @"Kolmnurk objekti ei ole tekitatud");
}

- (void)tearDown
{
    // Put teardown code here. This method is called after the invocation of each test method in the class.
    kolmnurk=nil;
    if(kolmnurk){

        XCTFail(@"Objekti ei hävitatud");
    }
    [super tearDown];
}

-(void)testKolmnurgaYmbermoot{
    //@3 märgitakse NSNumber tüüpi numbreid.
    XCTAssertEqualObjects(@3, [kolmnurk kolmnurgaPindalaArvutamineKorguse:2 Alus:3],
@"Kolmnurga pindala on vale");
    XCTAssertNotEqualObjects(@3,[kolmnurk kolmnurgaPindalaArvutamineKorguse:3 Alus:3] ,
@"Kolmnurga pindala ei tohiks olla 3");
}
-(void)testKolmnurgaPindala{

    XCTAssertEqual(12, [kolmnurk kolmnurgaYmbermoot],@"Kolmnurga ümbermõõt on vale");

}
```

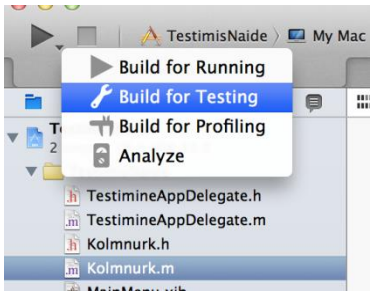
**Koodinäide 34. Kolmnurk klassi meetodite test klass**

Esmalt kaasame oma test klassi **Kolmnurk.h** faili, et me saaksime Kolmnurk klassi kasutada oma test klassis. Kõik test meetodid algavad XCT eesliitega.
Kontrollime **XCTAssertNotNil**'iga **setUp** meetodis, kas Kolmnurk objekt on tekitatud. Kui ei ole antakse veateade. Lisaks kontrollime kas kolmnurk objekt eemaldatakse mälust.

Esimeses meetodis võrrdleme, kas meie meetod annab vastavate sisendite korral õiged vastused võrreldes mõlema **NSNumber** objekti väärtusi, kasutades selleks meetodit **XCTAssertEqualObjects**. NSNumber numbreid märgitakse **@** sümboliga, millele järgneb number.

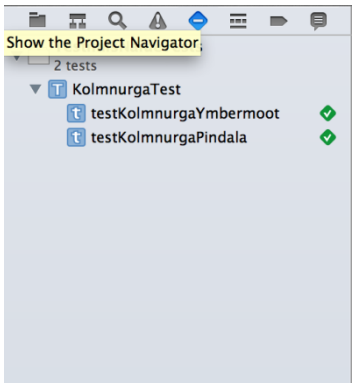
Teises meetodis kontrollime samuti, kas kolmnurga pindala on õigesti arvutatud, kuid **XCTAssertEqual** meetodiga, kuna objekte seal ei ole.

Testide järgu (ingl.k *build*) loomise käivitamiseks, hoitke hiirt all *Run* nuppu vajutades ja valige ilmnenud valikute aknast *Build for Testing* (vt Joonis 30).




Joonis 30. Testide järgu loomise käivitamine

Juhul kui test on ilma vigadeta testi läbinud, ilmneb testi kõrvale ikoon  ja testi ebaõnnestumisel  (vt Joonis 31. Testi navigaatoris olevate testide tulemused ).




Joonis 31. Testi navigaatoris olevate testide tulemused

Juhul kui testimise käigus esines viga on võimalik testi navigaatoris ebaõnnestunud testile klikkides näha täpselt kohta kus test ebaõnnestus. Täpsemat infot erinevate **XCTAssertion** juhtumite kohta saab addressilt:http://appleprogramming.com/blog/2013/12/26/xctest-assertions-documentation/.

## Ülesanded

1. Looge kalkulaator, mille abil saaks kasutaja korrutada, liita, lahutada ja jagada ning sooritatud tehteid koos vastustega andmebaasi salvestada.
2. Lisage esimeses ülesandes loodud rakendusele funktsionaalsus, mille abil juba sooritatud tehet ei tehta vaid kontrollitakse andmebaasist, kas vastav tehe on olemas. Juhul kui vastav tehe on olemas, kuvatakse vastava tehte vastus kasutajale.
3. Lisage testid, mis kontrollivad, kas teie meetodid arvutavad õigesti.
4. Muutke kalkulaatori rakendust nii, et see võtab andmebaasist juhusliku tehte koos vastusega ja kuvab tehte kasutajale ning kontrollib, kas kasutaja on vastavale tehtele andnud õige vastuse. Kasutajale kuvatakse ka tabelisse tema antud vastuste ajalugu (tabelis võiks kuvada tehte ja kas vastaja andis õige või vale vastus).

## Kasutatud kirjandus


Apple Inc. (2013). *Mac Developer Library* . Kasutamise kuupäev: 20. March 2014. a., allikas Apple Inc.: https://developer.apple.com/library/mac/navigation/index.html

Apple Inc. (2007). *AppleScript Overview*. Kasutamise kuupäev: 22. 04 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/applescript/Conceptual/AppleScriptX/AppleScriptX.html

Apple Inc. (2007). *NSButton Class Reference*. Kasutamise kuupäev: 1. 05 2014. a., allikas Mac Devloper Library: https://developer.apple.com/library/mac/documentation/cocoa/reference/applicationkit/classes/NSButton_Class/Reference/Reference.html

Apple Inc. (2012). *About Objective-C*. Kasutamise kuupäev: 01. 04 2014. a., allikas Mac Developer Libray: https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

Apple Inc. (2012). *Mac Developer Library*. Kasutamise kuupäev: 14. 04 2014. a., allikas Key-Value Coding Programming Guide: https://developer.apple.com/librarY/mac/documentation/Cocoa/Conceptual/KeyValueCoding/Articles/Overview.html#//apple_ref/doc/uid/20001838-SW1

Apple Inc. (2012). *NSTextField Class Reference*. Kasutamise kuupäev: 1. 05 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/cocoa/reference/applicationkit/classes/NSTextField_Class/Reference/Reference.html

Apple Inc. (2012). *Outlets*. Kasutamise kuupäev: 19. 03 2014. a., allikas iOS Developer Library: https://developer.apple.com/library/ios/documentation/general/conceptual/CocoaEncyclopedia/Outlets/Outlets.html

Apple Inc. (2013). *NSNumber Class Reference*. Kasutamise kuupäev: 01. 05 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/cocoa/reference/foundation/classes/nsnumber_class/Reference/Reference.html

Apple Inc. (2013). *NSObject Class Reference*. Kasutamise kuupäev: 01. 05 2014. a., allikas Mac Developer Library:

https://developer.apple.com/library/mac/documentation/cocoa/reference/foundation/classes/nsobject_class/reference/reference.html

Apple Inc. (2013). *Sprite Kit Framework Reference*. Kasutamise kuupäev: 22. 04 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/SpriteKit/Reference/SpriteKitFramework_Ref/_index.html

Apple Inc. (2013). *Transitioning to ARC Release Notes*. Kasutamise kuupäev: 01. 04 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html

Apple Inc. (2014). *Cocoa Frameworks*. Kasutamise kuupäev: 14. 03 2014. a., allikas Developer: https://developer.apple.com/technologies/mac/cocoa.html

Apple Inc. (2014). *NSString Class Reference*. Kasutamise kuupäev: 05. 01 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/Reference/NSString.html

Apple Inc. (2014). *NSTableView Class Reference*. Kasutamise kuupäev: 01. 05 2014. a., allikas Mac Developer Library: https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ApplicationKit/Classes/NSTableView_Class/Reference/Reference.html

Big Nerd Ranch. (2014). *Objectivec programming the big nerd ranch guide*. Kasutamise kuupäev: 16. 03 2014. a., allikas Big Nerd Ranch: https://www.bignerdranch.com/book/objectivec_programming_the_big_nerd_ranch_guide

*CocoaPods*. (2014). Kasutamise kuupäev: 15. 04 2014. a., allikas http://cocoapods.org/

*Delegates and Data Sources*. (2012). Kasutamise kuupäev: 22. 03 2014. a., allikas iOS Developer Library: https://developer.apple.com/library/ios/documentation/general/conceptual/CocoaEncyclopedia/DelegatesandDataSources/DelegatesandDataSources.html

Hawkins, J. (2011). *Cocoa and Objective-C Cookbook.* Birmingham: Packt Publishing.

Hillegass, A., & Ward, M. (2013). *Objective-C Programming: The Big Nerd Ranch Guide (2nd Edition).* Big Nerd Ranch Guides.

Kippar, J. (2013). *.Net raamistik.* Kasutamise kuupäev: 20. Aprill 2014. a., allikas Kursuseprogramm: http://minitorn.tlu.ee/~jaagup/kool/java/kursused/13/dotnet/juht.html

Outercurve Foundation . (2014). Kasutamise kuupäev: 01. 05 2014. a., allikas NuGet: http://www.nuget.org/

Poola, K. (6. September 2010. a.). Sissejuhatus Apple-i mobiilsetele tehnoloogiatele. Kasutamise kuupäev: 14. Aprill 2014. a., allikas Õppekava aine: https://itcollege.ois.ee/et/curriculum-subject/view?curriculum_id=3&subject_id=213&year=2013

Vallaste, H. (2014). *e-teatmik - IT ja sidetehnika seletav sõnaraamat*. Kasutamise kuupäev: 1. 05 2014. a., allikas Vallaste: http://www.vallaste.ee/

Wikipedia. (2013). *Foundation Kit*. Kasutamise kuupäev: 14. 03 2014. a., allikas Wikipedia: http://en.wikipedia.org/wiki/Foundation_Kit

Villems, A., Kusmin, M., Peets, M.-L., Plank, T., Puusaar, M., Pilt, L., et al. (2012). *Juhend kvaliteetse õpiobjekti loomiseks.* Tallinn: Eesti Infotehnoloogia Sihtasutus.