Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Porovnání programovacích jazyků na platformě Apple

Tomáš Hamerník

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Hamerník

Informatika

Název práce

Porovnání programovacích jazyků na platformě Apple.

Název anglicky

Programming languages comparison on the Apple platform.

Cíle práce

Tato práce popíše praktickou implementaci myšlenky objektového programování na mobilní platformě Apple, operační systém iOS a především porovnání jazyka Objective-C a Swift, které bude demonstrováno na příkladu malé aplikace.

Metodika

Prvním krokem pro vytvoření teoretického podkladu je znalost historického vývoje systému iOS, jeho jazyků a vlastností. Dále bude nutné nastudovat potřebnou literaturu k pochopení základních teorií objektového modelování a tvorby aplikací v prostředí Xcode 6.0. Tyto předpoklady umožní tvorbu praktické části práce, kde se budeme zabývat porovnání syntaxe a dalších rozdílů mezi jazykem Objective-C a Swift, které budou demonstrovány pomocí malé aplikace.

Doporučený rozsah práce

40 stran

Klíčová slova

programování mobilních zařízení, platforma Apple, iOS, Swift

Doporučené zdroje informací

HILLEGASS, A.; WARD, M. Objective-C Programming: The Big Nerd Ranch Guide. Atlanta: Big Nerd Ranch Guides, 2013. 325s. ISBN 978-0321942067.

KEUR, C.; HILLEGASS, A.; CONWAY, J. iOS Programming: The Big Nerd Ranch Guide. Atlanta: Big Nerd Ranch Guides, 2014. 560s. ISBN 978-0321942050.

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 7. 3. 2015

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 02. 03. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Porovnání programovacích jazyků na platformě Apple" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 10.3.2016 _____

Poděkování

Rád bych touto cestou poděkoval panu doc. Ing Vojtěchu Merunkovi, Ph.D. za odbornou pomoc, konzultaci a vedení práce.

Porovnání programovacích jazyků na platformě Apple.

Souhrn

Tato práce pojednává o porovnání programovacích jazyků na platformě Apple, přesněji na operačním systému iOS. Hlavním tématem je porovnání jazyků pro dané prostředí, kterými jsou Objective-C a Swift.

V teoretické části práce je vysvětlena základní terminologie objektového programování a popsán operační systém iOS. Dále je rozebráno vývojové studio Xcode a programovací jazyky Objective-C a Swift.

Praktická část obsahuje návrh a vytvoření ukázkového programu, jeho následnou dokumentaci a popis jednotlivých částí zdrojového kódu pro oba programovací jazyky.

Klíčová slova: Platforma Apple, iOS, Swift, programování mobilních zařízení.

Programming languages comparison on the Apple platform

Summary

This bachelor thesis deals with issues of programming languages comparison on the Apple platform, more exactly on the operating system iOS. Main topic is comparison of appropriate languages, which are Objective-C and Swift.

In the first section of the bachelor thesis is clarified basic terminology of object oriented programming and description of operating system iOS. This is followed by description of programming studio Xcode and languages Objective-C and Swift.

Practical section of the bachelor thesis is focused on design, creation and description of the model application. Code description is performed through comments of the essential parts of the code for both programming languages.

Keywords: Apple platform, iOS, Swift, mobile devices programming.

Obsah

Seznam obrázků

1 Úvod

Již nějakou dobu má společnost Apple se svými iOS produkty velký podíl na trhu a těší se neobyčejné popularitě. Pro tato zařízení psali aplikace ve svém vytvořeném jazyce Objective-C. Tím, jak se postupem času upravoval, začal být komplikovanější a méně přehledný. V roce 2014 proto představili zcela nový programovací jazyk Swift, který se stal volně dostupný společně s vývojovým prostředím Xcode 6.0. Tyto programovací jazyky jsou v jedné aplikaci zcela kompatibilní. Aplikace může být psána oběma jazyky najednou, nebo každým zvlášť. V tuto dobu ještě není zcela jasné, zda nový Swift zcela nahradí původní Objective-C. Proto je zcela aktuálním tématem jejích porovnání a testování.

2 Cíl práce a metodika

2.1 Cíl práce

Tato práce popisuje základní myšlenky objektového programování na platformě Apple, operační systém iOS a především jeho programovací jazyky Objective-C a Swift, kde jejich rozdíly budou popsány a demonstrovány na dvou naprosto identicky fungujících malých aplikacích, přičemž aplikace jsou napsány v rozdílných jazycích.

2.2 Metodika

Základem je vytvoření teoretické znalosti o systému iOS a jeho jazyků. Dále je nutné nastudovat potřebnou literaturu k pochopení základních teorií objektového programování. Poté je potřeba ovládnout funkcionalitu prostředí Xcode. Tyto znalosti umožňují postup k praktické části práce, kde se budeme zabývat porovnáním syntaxe a dalších rozdílů mezi jazykem Objective-C a Swift. Tyto rozdíly budou demonstrovány a popsány na ukázce malé aplikace.

I. Teoretická východiska

3 Objektové programování

Objektově orientované programování (dále jen OOP) je nástroj pro modelování dat a tvorbu softwaru (Merunka, 2008, s. 24). Tato kapitola pojednává o historii, základních pojmech a principech.

3.1 Počátky vývoje objektově orientovaného programování

Počátky OOP sahají až do 60. let 20. století. Prvním systémem, který měl jeho rysy, byl již programovací jazyk Simula, který pochází z roku 1967. OOP jako takový, byl vytvořen v průběhu 70. let v USA. Přesněji ve výzkumném středisku Palo Alto Research Center v Kalifornii.

Hlavní dva týmy, které se zasloužily o tvorbu objektově orientovaného přístupu, byly týmy vedené Alanem Keyem a Adelou Goldbergovou. Jejich projekt Dynabook byl zaměřen na vývoj osobního počítače budoucnosti. Předpokládalo se, že softwarové prostředí počítače bude plnit úlohu operačního systému i programovacího jazyka s jeho prostředím. Tento software byl pojmenován Smalltalk.

Po ukončení projektu se část týmu Adely Goldbergové soustředila na další vývoj jazyka Smalltalk. Jiní spolu s Alanem Keyem přešli do firmy Apple computers. Zde se podíleli na vývoji prvního osobního počítače Lisa, založeného na myšlenkách Dynabooku (Merunka, 2008, s. 23, 24).

3.2 Základní obecné pojmy a přístupy

V této části bude vysvětlena základní terminologie objektově orientovaného přístupu. Tyto přístupy ovšem nejsou hlavním principem OOP. Tím principem je především *volná síť vzájemně komunikujících objektů* (Čada, 2009b, s. 20).

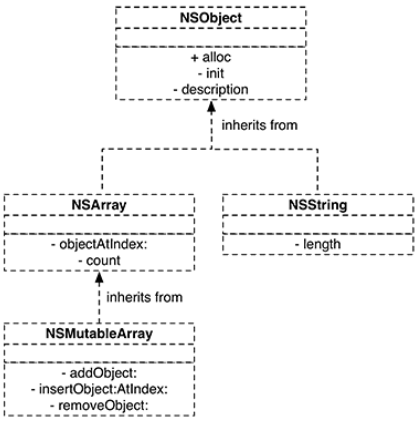
3.2.1 Základní myšlenka Objektově orientovaného programování

Jak již bylo zmíněno v kapitole 3.2, OOP je postaveno na principu vzájemné komunikace objektů. *Objekt* je základní jednotka objektového programování, která představuje model reálného světa. V objektu jsou obsažena veškerá data, chování a vlastnosti těchto dat. Objekt má jednotlivé atributy (vlastnosti), které ho definují. Díky tomu může popisovat

živé, neživé, ale i nehmotné a abstraktní entity. *Objekty jsou spolu propojeny vazbami a navzájem na sebe působí* (komunikují) (Merunka, 2008, s. 25).

3.2.2 Třídy

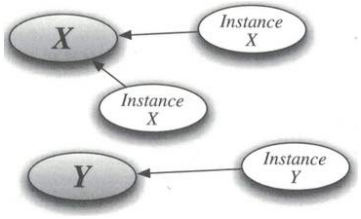
Třída je označení pro objekt, avšak ne vždy, který reprezentuje určitý druh objektů. Tedy vlastnosti a metody objektů, které pod danou třídu náleží. Tyto objekty obsahují odkaz na svou třídu a vlastnosti z ní přebírají pomocí dědičnosti (viz. kapitola 3.2.5). Tvoří tak postupnou hierarchii viz. obrázek 1.



Obrázek 1 - Hierarchie tříd (Keur a kol., 2014)

Třídy samy jsou tedy objekty obsahující vlastnosti a metody. Objekty, které je přebírají se nazývají *instance třídy* a nejsou třídami.

- *Definice třídy podle O. Čady:* třída je prvek objektového počítačového systému, který odpovídá „druhu objektu“. U každého objektu je zřejmé, které třídě náleží, a mezi kterými třídami je vhodným způsobem definovaná hierarchie, postihující jejich vzájemné podobnosti (Čada, 2009b).



Obrázek 2 - Třídy a jejich instance, kde třídy jsou objekty (Čada, 2009b, s. 27)

Instance mají vlastní *instanční proměnné* a také *metody*. Instanční proměnné obsahují konkrétní hodnoty a metody jsou procedury, které pracují na základě hodnot a určují tak chování objektu (Čada, 2009b, s. 27).

3.2.3 Zprávy

Jednou z vlastností objektů je schopnost reagovat na požadavky. Tyto požadavky jsou označovány jako *zprávy*. Objekty jsou tedy příjemci zpráv nebo odesílatele zpráv (Merunka, 2008, s. 26). Zprávy tedy spouští vybranou metodu (viz. kapitola 3.2.4), přičemž dotazovaný objekt rozhoduje, zda bude požadavek proveden, pozměněn nebo odmítnut.

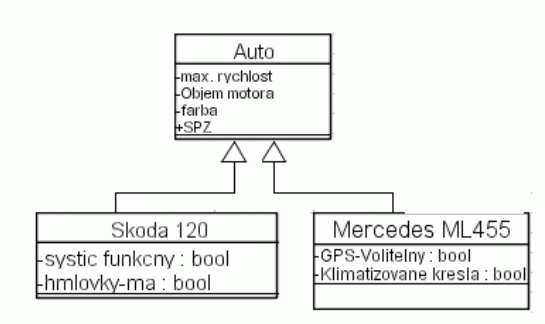
- *Definice zprávy:* Zpráva je obecný požadavek na nějakou činnost, který lze předat objektu. Ten na ni obvykle, ale ne nutně, reaguje vyvoláním vhodné metody, a to podle vlastního rozhodnutí (Čada, 2009b, s. 30).

3.2.4 Metody

Metody jsou řídící struktury, které jsou vyvolány zprávou. Metody definují chování objektů a pracují s jejich daty. Provádějí výpočty a procesy a umožňují přistupovat k datům. Metody mohou být definovány jak ve třídách, tak i v jejich potomcích. Každá metoda může mít upravená přístupová práva. Tato práva definují, jaké objekty mohou dotyčnou metodu volat. Metody potomků mohou být upravené. Předek určí, že metoda existuje, ale potomek nastaví konkrétní strukturu metody. Každý potomek tedy může mít rozdílný výstup při stejných vstupech (Čada, 2009b, s. 27).

3.2.5 Dědičnost

Vlastnost umožňující objektům dědit vlastnosti ze své nadtřídy. Tvoří se tak hierarchie. Při vytváření třídy může programátor vybrat nadtřídu, která poskytne veškeré své vlastnosti. Tyto vlastnosti lze dále rozšiřovat podle potřeby. Metody nadtřídy mohou být nahrazeny metodami novými. Upravování podtříd má za následek větší konkrétnost.



Obrázek 3 – Dědičnost (Čižmár, 2005)

Tato vlastnost OOP je velice praktická, jelikož v objektových jazycích je spousta tříd se svými užitečnými metodami již předdefinována. Je tak velice jednoduché je využívat při tvorbě vlastních komplexnějších metod (Fojtík, 2002).

Typy dědičnosti:

- *Jednoduchá* – Třída má jednoho předka. Postupně se tak tvoří strom zobrazující hierarchii.
- *Vícenásobná* – Třída má více než jednoho předka.

3.2.6 Zapouzdření

Elementární vlastnost objektového programování. Zajišťuje neviditelnost metod a atributů uvnitř objektu pro ostatní objekty. Tyto přistupující objekty vidí pouze chování a mohou přistupovat pouze k vybraným metodám a atributům, které mají specificky nastavená práva. Pro instanční proměnné, které mají být přístupné zvenčí, je třeba nastavit přístupové metody zvané *accessors*. Těmito přístupovými metodami jsou *getter*, který získává hodnoty instanční proměnné a *setter*, který ji nastavuje (Čada, 2009b, s. 33).

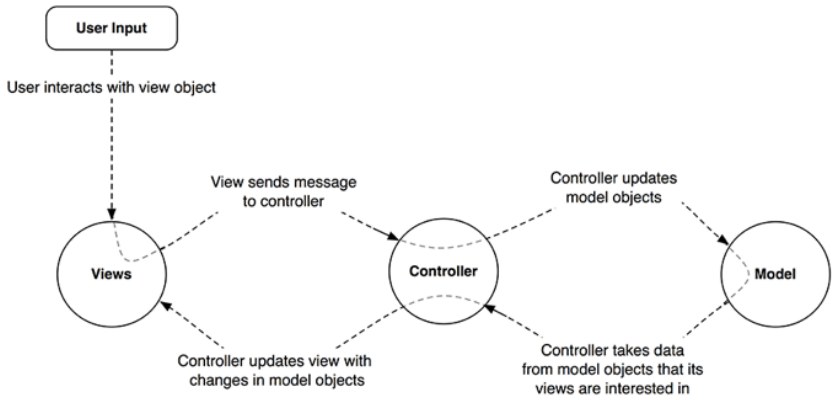
3.2.7 Polymorfismus

Vlastnost dvou objektů na stejný podnět reagovat různě. V praxi to znamená přepsat metodu nadřazeného objektu objektem podřazeným (například `viewDidLoad` metoda viz. později) (Bayer, 2015).

3.2.8 Model – View – Controller

Základní návrhový vzor aplikace ve zkratce nazývaný MVC. V tomto principu je každý objekt buď *modelem*, *view* nebo *controller*.

- *View objekty* – Jsou viditelné pro uživatele. Příklady těchto objektů mohou být popisky, textová pole, posuvníky a další prvky, které tvoří uživatelské prostředí.
- *Model objekty* – Mají na starost skladovat a zpracovávat data jako taková. Nestarají se o implementaci dat. Tyto objekty nevědí nic o uživatelském rozhraní.
- *Controller objekty* – Starají se o implementační logiku. Řídí, jaká data uživatel vidí skrze View objekt, přičemž tato data se přebírají z Model objektu. Například uživatel vybere položku a upraví ji. Tato informace je předána Controller objektu, který upraví data Model objektu. Controller poté vezme upravená data a předá je do View, kde uživatel vidí změnu viz. obrázek 4 (Keur a kol., 2014).



Obrázek 4 - MVC schéma (Keur a kol., 2014)

3.3 Shrnutí kapitoly

V úvodu kapitoly 3 byla stručně popsána historie vytvoření objektově orientovaných principů. Dále souhrnně popsány základní myšlenky objektového programování, které jsou neustále využívány v jazyce Objective-C a Swift. Byla také ujasněna základní terminologie jako jsou třídy, metody, zprávy.

4 iOS

Operační systém iOS provází mobilní zařízení od společnosti Apple Inc. již od představení první verze mobilního telefonu iPhone, tehdy pojmenovaný jako iPhone OS. Později v roce 2010 byl přejmenován na iOS s příchodem čtvrté verze. Nyní je využíván mobilními telefony iPhone, tablety iPad a zařízení iPod touch.

4.1 Historický přehled verzí

V následující části budou zjednodušeně představeny verze operačního systému iOS, pro získání odlehčeného pohledu na vývoj funkcionality a nástrojů jím používaných.

4.1.1 iPhone OS 1.x

Původní operační systém byl při představení prvního mobilního telefonu iPhone označen jako varianta OS X, než byl přejmenován. iPhone OS 1 byl jednoduchý intuitivní operační systém, obsahující pouze základní aplikace bez možnosti instalace jakýchkoliv rozšiřujících aplikací třetích stran. Za hlavní vlastnosti byl považován emailový klient, intuitivní softwarová klávesnice, internetový prohlížeč Safari upravený pro mobilní

zařízení, hudební přehrávač, kalendář, mapy, fotografie a především multidotykové ovládání systému (Alza, 2015a; Buster, 2013).

4.1.2 iPhone OS 2.x

Novější verze operačního systému iPhone OS byla představena společně s telefonem iPhone 3G v roce 2008. Veškeré zařízení s původním systémem bylo možné upgradovat na novější verzi. iPhone OS 2.x vylepšoval stávající aplikace. Nejdůležitějším vylepšením ovšem byla integrace App Storu a s tím spojené podpory aplikací třetích stran (Alza, 2015a; Buster, 2013).

4.1.3 iPhone OS 3.x

Třetí verze operačního systému představená v roce 2009. Obsahovala nové možnosti jako MMS, funkci kopírování, vyjmutí a vkládání, spotlight (vyhledávání v systému) či možnost nahrávání videa. Další novinkou byly nákupy přímo v aplikacích nebo vyskakovací notifikace. Kromě těchto vylepšení obsahoval množinu online služeb nazvanou MobileMe (Find my iPhone, Adresář, Kalendář a další), která byla později nahrazena službou iCloud (Alza, 2015a; Ritchie 2009).

4.1.4 iOS 4.x

Poprvé byl označen operační systém pod názvem iOS až ve čtvrté generaci roku 2010. Hlavní novinkou byl omezený multitasking aplikací na pozadí. Do příchodu verze 4.0 se aplikace automaticky po stisknutí domovského tlačítka ukončily. Dalšími novinkami byla možnost video hovoru pomocí FaceTime aplikace, aplikace iBooks pro podporu elektronických knih a možnost tvorby složek na hlavní ploše (Alza, 2015b; Ritchie 2010).

4.1.5 iOS 5.x

Pátá generace operačního systému se proslavila především hlasovou asistentkou Siri, díky které je možné určité funkce ovládat hlasovými příkazy (nastavit budík, vyhledávat na internetu, přidat událost do kalendáře…). Poprvé bylo umožněno aktualizovat systém bez připojení k počítači pouze pomocí Wi-Fi. Přidáno bylo nové notifikační centrum a vylepšené byly aplikace Safari, nebo Camera, která nově umožňovala editaci fotografií (Alza, 2015b; Buster, 2013).

4.1.6 iOS 6.x

iOS verze 6 byl představen v roce 2012, přičemž hlavní vylepšením byla podpora 16:9 displejů jako příprava pro nadcházející verzi iPhone 5. Kromě dalších minoritních vylepšení obsahoval integraci aplikace Facebook a vylepšenou hlasovou asistentku Siri (Imore, 2015a).

4.1.7 iOS 7.x

Sedmá verze operačního systému iOS byla výrazná velkými grafickými změnami oproti předešlým verzím. Barvy se zvýraznily a design přesel do ostřejších tvarů. Kromě designových úprav iOS 7 obsahoval nové „Control Center“ pro rychlý přístup k nejdůležitějším vlastnostem (světlo, bluetooth, kamera…), iTunes radio, CarPlay a další (Buster, 2013).

Obrázek 5 - porovnání UI iOS 6.x a iOS 7.x (Sebastian, 2013)

4.1.8 iOS 8.x

Veze 8 byla představena společně se zařízením iPhone 6 v roce 2014. Obsahovala mnoho nových funkcí, které kooperují se zařízením Apple Watch, jako například aplikace Health. Pro aplikaci Health byl vytvořen speciální nástroj s názvem HealthKit, který umožňuje práci s aplikacemi zaměřenými na zdraví uživatele. Další nové aplikace byli iCloud Drive nebo Family sharing (Imore, 2015b).

4.1.9 iOS 9.x

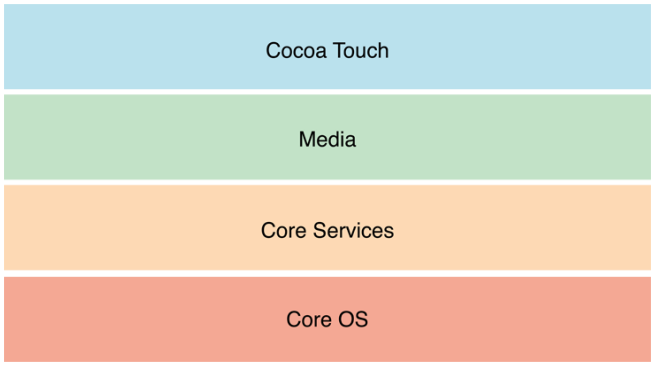
Nejnovější verze operačního systému, dostupná v září 2015, nepřinášela vizuální změny, ale spíše úpravy detailů. Systém jako takový zmenšil svou velikost a snížil spotřebu

energie zařízení. Velkou inovací byla možnost Split – screen pro iPad Air 2 a iPad Pro. Tato funkce umožňuje běh dvou aplikací vedle sebe tak, že uživatel nemusí přes nabídku multitaskingu přecházet mezi aplikacemi. Zvyšuje tak produktivitu práce. Základní balík aplikací provázely také inovace. Především aplikaci poznámek, fotografií nebo map, které mají pro určitá města integrovanou městskou hromadnou dopravu (Imore, 2015c; Svatoš, Pultzner 2015; Appledystopia, 2015).

4.2 Architektura systému

Operační systém iOS používá vrstvenou architekturu. Zde iOS figuruje jako prostředník mezi hardwarem zařízení a aplikacemi. Díky této vlastnosti se aplikace nemusí starat o ovládání hardwarových komponent, které jsou na různých zařízeních odlišné.

Implementace technologií iOS je dělena do vrstev, které jsou zobrazeny na Obrázku 6. Nižší úrovně obsahují základní technologie a služby. Vyšší jsou postaveny na technologiích nižších vrstev a tím se stávají sofistikovanější (Apple, 2015a).



Obrázek 6 - Vrstvy iOS (Apple 2015a)

4.2.1 Cocoa Touch Layer

Nejvyšší vrstva operačního systému iOS obsahuje *frameworky* (knihovny), které tvoří vzhled výsledné aplikace, základní infrastrukturu a podporuje klíčové funkce (Apple, 2015c).

- Příklady základních funkcí
    - *Multitasking:* Po stisku domovského tlačítka jsou aplikace přepnuty na pozadí. Pokud nevykonávají činnost jsou uvedeny do stavu *freez-dried*. Zůstanou zavedené v paměti zařízení, ale nepřekládá se žádný kód.
    - *TextKit:* Sada tříd pro práci s textem a jeho typologií.

- *Auto Layout:* Funkce umožňující tvořit dynamické uživatelské prostředí aplikací s minimem psaného kódu. Definuje pravidla, jak vzdálené mají být objekty jeden od druhého (*constrains*).

- Cocoa Touch základní technologie (frameworky)
  - *GameKit Framework:* Podpora Game Center pro interakci uživatelů hrající hry pro více hráčů.
  - *MapKit:* Umožnění implementace map do aplikace a jejich modifikace, určování polohy a další.
  - *UIKit Framework:* Poskytuje elementární nástroje pro implementování grafických a událostmi řízených aplikací. Umožňuje například práci se základními objekty jako jsou textové pole, tlačítka, ale také ovládání fontů a barev. Dále umožňuje pracovat s dotyky uživatele, senzory a kamerou, nebo *cut*, *copy* a *paste* funkce.

4.2.2 Media Layer

Takzvaná mediální vrstva obsahuje nástroje pro práci s grafickými, hudebními a video komponenty. V následující části budou uvedeny pouze vybrané ukázky jednotlivých mediálních komponent (Apple, 2015g).

- Grafické technologie (frameworky)
  - *Core Graphics framework:* Nástroj pro kresbu 2D vektorových obrazců, renderování obrázků a gradientů.
  - *Core Animation:* Součást nástroje *Core graphics Framework*, umožňující práci a optimalizaci animací aplikace. *UIKit* používá *Core Animation* pro zobrazování animací v jednotlivých pohledech. Pro úpravu chování animace je ale nutno pracovat přímo s tímto nástrojem.
  - *OpenGL ES:* Umožňuje 2D a 3D rendrování výstupu. Používán především pro tvorbu her.
- Audio technologie (frameworky)
  - *Media Player Framework:* Poskytuje přístup do knihovny iTunes. Skrze tento *framework* lze jednoduše propojit audio obsah s tvořenou aplikací.
  - *AV Foundation:* Sada funkcí pro záznam a nahrávání zvukového i obrazového vstupu.

- *Core Audio:* Pokročilý nástroj pro práci s jednotlivými audio komponenty zařízení. Definuje formáty jednotlivých nahrávek.

- Video technologie (frameworky)
  - *AVKit:* Nástroj obsahující funkce pro práci s *fullscreen* (zobrazení přes celou obrazovku) i částečným zobrazením video obsahu a podporu dalších přehrávacích funkcí pro uživatele.
  - *Core Media:* Obsahuje definice datových typů a rozhraní pro manipulaci s media daty. Ve většině případů aplikace nepřistupují k tomuto *frameworku* přímo.

4.2.3 Core Service Layer

V této vrstvě lze nalézt elementární systémové služby. Ty nejdůležitější obsahují nástroje *Core Foundation* a *Foundation Framework*. Zde také leží technologie podporující síťovou komunikaci, iCloud, sociální média a lokalizaci (Apple, 2015e).

- Vybrané nástroje Core Service Layer
  - *CDNetwork Framework:* Obsluha síťových protokolů.
  - *Core Foundation Framework:* Nástroj založený na jazyce C poskytuje elementární správu dat a služeb iOS, jako jsou například datové typy kolekcí (pole), správa typu `string`, správa data a času, management vláken a cyklů a mnoho dalšího.
  - *Foundation Framework:* Obsahuje stejnou funkcionalitu jako *Core Foundation Framework*, ale poskytuje funkce v jazyce Objective-C.
  - *HealthKit Framework:* Nový specifický nástroj vytvořený pro zařízení a aplikace (*Health*) sledující zdravotní údaje uživatele. Poskytuje možnost jednoduše sledovat a ukládat tyto data.

4.2.4 Core OS Layer

Nejnižší vrstva na níž jsou ostatní postaveny. Ačkoliv k ní programátor málokdy přistupuje, je používaná vrstvami vyššími. Obsahuje nástroje pro správu bezpečnosti komunikace a přímou interakci s externími hardwarovými komponenty (Apple 2015d).

- *Accelerate Framework:* Obsahuje nástroje pro práci s digitálními signály.

- *Security Framework:* Poskytuje obecné bezpečnostní nástroje pro zaručení bezpečnosti dat. Dále poskytuje nástroje pro práci s certifikáty, klíči a nastavení důvěrnosti a další.
- *Systém:* Poskytuje *drivery* a nízko úrovňové UNIX rozhraní. Stará se o správu paměti, vláken, souborového systému, sítí. Z bezpečnostních důvodů je možné přistupovat k jádru a driverům pouze skrze několik vybraných *frameworků*.

4.3 Shrnutí kapitoly

Obsahem kapitoly bylo nejprve stručné představení verzí operačního systému iOS. Již podle těchto zjednodušených popisů je zřejmé, jak se musel programovací jazyk a jeho nástroje vyvíjet a přidávat nové možnosti pro práci s novějšími prvky jednotlivých verzí iOS. Druhá část byla zaměřena na architekturu vrstev operačního systému a následně na popis jejich základních nástrojů, který poskytuje odlehčený náhled na fundamentální možnosti.

5 Xcode

Xcode v současné verzi 7.x je vývojovým prostředím pro platformu Apple jak pro OS X tak pro operační systém iOS. Vývojové prostředí obsahuje balíček vlastností, které jsou klíčové pro tvorbu aplikací. Kromě těchto vlastností také obsahuje celou dokumentaci obou programovacích jazyků (třídy, metody...). První verze prostředí Xcode, byla vydána v roce 2003. Následující kapitola je z části založena na subjektivních zkušenostech získaných z manuálů a příruček.

5.1 Základní vlastnosti Xcode

V následující kapitole jsou jednotlivě popsány základní komponenty vývojového prostředí Xcode.

5.1.1 LLVM – kompilátor

LLVM kompilátor byl poprvé představen v Xcode 3.2. Je založený na open source projektu LLVM.org, který byl započat na univerzitě v Illinois a slouží pro generaci zdrojového kódu. LLVM je balíček knihoven. Tyto knihovny jsou, mnohem lépe upravovatelné a umožňují pružnou inovaci při řešení problémů. Od Xcode 5.0 nahradil původní kompilátor GCC.

Jeho hlavní součástí pro účely této práce je takzvaný *Clang*, což je kompilátor pro jazyky C, C++ a Objective-C podporovaný především společností Apple. Pomocí *Clang* je generován mezikód a následně zdrojový kód jako takový. *Clang* je zaměřený především na urychlování překladu kódu oproti původnímu překladači GCC. Jeho vlastností je také vyhledávání a vyznačování chyb a výstražné znamení upozorňující na možné chyby (LLVM, 2015a; Apple, 2015f).

5.1.2 LLDB debugger

LLDB je debugger implementovaný v prostředí Xcode. Obsahuje komponenty, které kooperují s knihovnami LLVM překladače (LLVM, 2015b)

5.1.3 Simulátor

Zabudovaná aplikace prostředí Xcode, která umožňuje okamžité testování výstupu aplikace a její chování tak, jak by se chovala na reálném zařízení. Je to bezpečné testování jak pro iOS zařízení, tak pro OS X i Apple Watch. Simulátor může být nastaven podle verze zařízení. Pro iOS zařízení to může být iPhone od vrze 4S a iPad od verze 2.

Vlastnosti simulátoru jsou téměř identické jako na zařízení od doteků přes otáčení obrazovky, tahy, notifikace až po testování uměle vyvolaných chyb a zjištění přesného chování testované aplikace.

5.1.4 Instruments

Kromě samotného vytvoření programu a zajištění funkcionality požadovaných vlastnosti, je pro vývojáře důležité ladit vytvořený produkt. K účelu nalezení chyb je v Xcode balíček nástrojů *Instruments*. Pomocí těchto aplikací lze zjišťovat chování aplikací a jejich případné chyby. Základními nástroji jsou *Time Profiler*, *Allocations* a další (Frost, 2015).

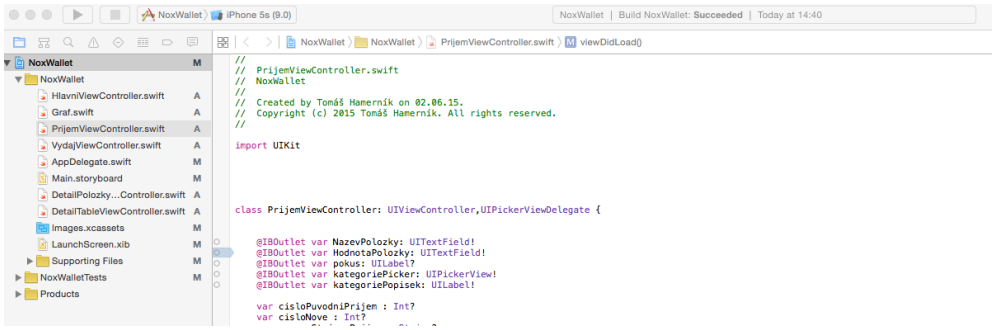
5.1.5 Interface builder

Interface builder je základní aparát pro tvorbu grafického rozhraní aplikace. Programátor zde utváří jednotlivé pohledy (scény) a jejích propojení pomocí vazeb mezi objekty (tlačítka, posuvníky a další). Tyto pohledy jsou přiřazeny k vytvořené třídě, která je podtřídou základní vyšší třídy a tím má definované vlastnosti. Objekty na scéně (tlačítka, textová pole, popisky) jsou skrze *interface builder* napojeny na třídu dané scény, kde je implementováno chování těchto objektů. Souhrnná nastavení zvaná *Utilities* jsou efektivní cestou pro nastavení jednotlivých prvků v obrazovkách nebo obrazovky samotné.

5.1.6 Editor zdrojového kódu

Po vytvoření uživatelského prostředí pomocí *Interface builderu* popsaného v kapitole 5.1.5, je nutné obrátit pozornost na editor zdrojového kódu. Skrze tento nástroj vývojář tvoří logiku samotné aplikace pomocí programovacího jazyka (v tomto případě Objective-C nebo Swift).

Před tvorbou aplikace a v jejím průběhu je nutné vybrat typ souboru, do kterého se bude daná třída implementovat. Podle tohoto souboru překladač očekává syntaxi daného jazyka, kterou pomáhá dodržovat již v editoru zdrojového kódu. Tato pomoc spočívá v rozpoznání syntaktických, ale částečně i logických chyb a upozornění na ně.



Obrázek 7 - Editor zdrojového kódu

5.2 Shrnutí kapitoly

V kapitole byly souhrnně popsány základní části programovacího studia Xcode, které je základním kamenem tvorby aplikací pro platformu Apple.

6 Objective-C

Tato kapitola se zabývá základními informacemi o jazyku Objective-C 2.0. Počínaje historickou vsuvkou po zhodnocení základních vlastností a ukázkou elementárních struktur jako je tvorba metod a další.

6.1 Historický vývoj

Objektově orientovaný jazyk Objective-C byl navržen na počátku 80. let 20. století od Brand J. Coxe. Jazyk byl navržen jako objektová nadstavba jazyka C s přidanými prvky jazyka SmallTalk-80. Díky tomu byla umožněna práce s objekty. V původní verzi byla k dispozici jediná třída Objekt, která se starala o vytváření objektů, jejich zánik a posílání zpráv.

Roku 1988 společnost NeXT licencovala Objective-C a vytvořila jeho knihovny a vývojové prostředí NextStep. Později v roce 1994 bylo vydáno vývojové prostředí OpenStep, na kterém se podíleli společnosti NeXT a Sun Microsystems. V této době byly přidány některé knihovny, které se používají i dnes (čítač referencí, poloautomatická správa paměti). Zajímavým faktem je prefix NS, který se používá u tříd v Objective–C, kde písmeno „S“ značí Sun (jméno jedné z firem), přičemž tento prefix zůstal dodnes.

Objective-C měl zajištěnou existenci hlavně díky akvizici společnosti NeXT společností Apple Computers. Ta roku 1996 přebrala NextStep / OpenStep, které se stalo základem operačního systému Apple OS X, jež se vyvíjí i dnes. Knihovnu frameworků podporující Objective-C přejmenovala společnost Apple na Cocoa.

V roce 2007 nastal další milník pro tento jazyk. Spolu s novým zařízením iPhone byla vydána verze Objective-C 2.0. Později na nátlak veřejnosti byl dovolen vývoj nativních aplikací třetích stran na iPhone a vydána vývojářská sada nástrojů SDK (Sowftware Development Kit) (Čada, 2009a, s. 14; Kochan, 2010, s. 13).

6.2 Základní vlastnosti jazyka

Jak je již z názvu zřejmé, Objective-C je nadstavbou jazyka C. Tím pádem zde lze nalézt veškeré funkční vlastnosti a konstrukce jazyka C. Jazyk C a jeho základní vlastnosti nejsou předmětem této práce, a proto se jimi dále již nebudeme zabývat.

Objektová nadstavba byla převzata z jazyka SmallTalk jak již bylo zmíněno, a to především v podobě zasílání zpráv objektu pomocí hranatých závorek.

- *Příklad:* `[objekt zpráva];`

Hned po vytvoření projektu v jazyce Objective-C je zřejmá první klíčová vlastnost. Tou je vytvoření dvou hlavních typů souborů `.h` a `.m`.

Hlavičkový soubor – Zde se deklarují proměnné, struktury, zprávy (metody) a instanční proměnné (*property*), které mají být viditelné i v jiných souborech pokud daný hlavičkový soubor importujeme a vytvoříme instanci dané třídy. Tyto soubory mají koncovku `.h`.

- *Příklad:*

```
@interface NazevTridy : NazevNadtřídy
{
//proměnné;
}
//atributy (property)
//metody
@end
```

Implementační soubor – Zde se implementuje logika metod (stejnojmenné zprávy) navržených v hlavičkovém souboru. Používají se zde již předdefinované metody, objekty a jejich vlastností z daných tříd a implementovaných knihoven. Definuje jakým způsobem bude třída reagovat na přijaté zprávy (Demčák, 2010). Tyto soubory jsou značeny koncovkou `.m`.

- *Příklad:*

```
#import „NazevTridy.h“
@interface NazevTridy ()
@end
@implementation NazevTridy
//Implementace
@end
```

Další významnou vlastností je tvorba instančních proměnných od Objective-C objektů jako je `NSString`, `NSArray` a další. Na rozdíl od elementárních datových typů jako je `int` nebo `BOOL` musí být vytvořeny jako *pointer* (ukazatel). Z toho plyne, že instanční proměnná je ukazatel na místo v paměti, kde je její hodnota uložena. Pokud se tedy hodnoty mezi proměnnými předávají, pak se předává pouze ukazatel. To má za následek úsporu paměti a výpočetní kapacity.

V Objective-C je zpráva paměti z části spravována pomocí systému ARC (*Automatic Reference Counting*). Každý objekt uchovává informaci, kolik referencí (pointerů) je s ním asociováno. Pokud jsou všechny tyto reference ztraceny, objekt už není potřeba a je možné ho odstranit (Hillegass a kol., 2013). Základní syntaxe jazyka je vysvětlena v následující části kapitoly.

6.3 Základní syntaxe Objective-C

V této části kapitoly je představen popis nejpoužívanějších objektů a práce s nimi. Dále vysvětlena syntaxe zasílání zpráv, metod a *property* (instanční proměnné).

6.3.1 Import hlavičkových souborů

Vlastnost známá z jazyka C. Deklarované vlastnosti třídy v hlavičkovém souboru .h mohou být jednoduše přístupné v implementačním souboru sobě vlastním i jiných tříd. Řešení je importovat daný soubor v implementačním souboru třídy, u které chceme vlastnosti zpřístupnit pomocí syntaxe `#import „hlavickovySouvor.h“`. Poté je možné přistupovat k vlastnostem i v jiné třídě pomocí instance třídy, ze které vlastnosti požadujeme.

- *Příklad:*

```
TridePristupu *instance = [[TridaPristupu alloc]init];
[instance zpráva];
```

6.3.2 Nejpoužívanější třídy

Nejčastější třídy, které jsou dostupné v *Cocoa Touch frameworku*. Používání těchto tříd je stěžejní pro jakoukoliv implementaci. V praxi je nutné využívat celou řadu dalších tříd, přičemž zde jsou vyjmenovány pouze vybrané elementární (Demčák, 2010).

- `NSString`: Třída pro nakládání s neměnným textovým řetězcem. Podtřídou je `NSMutableString` pro práci s proměnnými textovými řetězci.
- `NSNumber`: Třída pro práci s čísly.
- `NSArray`: Třída pro práci s polem. Podtřídou je mu `NSMutableArray`, který umožňuje pracovat s proměnným polem objektů. K objektu v poli se přistupuje pomocí jeho pozice (idexu) v poli.
- `NSDictionary`: Třída pro práci se sadou objektů vložených pod originálním klíčem. Díky tomuto klíči se pak k objektu přistupuje. Podtřídou je `NSMutableDictionary` pro proměnou množinu objektů v knihovně.
- `NSUserDefaults`: Třída pro ukládání nastavení objektů. K uloženým objektům se přistupuje pomocí klíče.
- `NSInteger`: Třída pro práci s celými čísly.

6.3.3 Property Objective-C

Instanční proměnné, anglicky *Property*, jsou definované v hlavičkovém souboru `.h`, nebo v implementačním souboru za značkou `@interface`. Tyto *property* pak mohou být viditelné ostatním souborům, kde je hlavičkový soubor implementován (pokud v něm jsou deklarovány) a vytvořená příslušná instance. Za *property* mohou být nastaveny třídy spravující data, jako je `NSArray`, ale i objekty, se kterými interaguje uživatel jako je například *textové pole.*

Hlavním účelem *property* je vytvořit přístupové metody *getter* a *setter*. Ve správném objektovém jazyce není možné přistupovat k objektu přímo, nýbrž skrze tyto metody. Po vytvoření *property* se tedy vytvoří metody, které umožňují získávat data z objektů a metody umožňující nastavení hodnot. *Property* jako takové mají následující atributy, které specifikují jejich chování:

- *Readonly:* vytvoří pouze metodu pro získání dat *getter*.
- *Strong:* Automaticky nastavená vlastnost. Vytváří takzvanou silnou vlastnící vazbu mezi *property* a přiřazenou hodnotou.
- *Weak:* Tvoří nevlastnický vztah a tak nebrání systému ARC zrušit referencovaný objekt. Používá se v případě existence možnosti ztráty hodnoty dané *property*.
- *Copy:* Vytvoří kopii přiřazené hodnoty místo odkazu na existující.
- *Nonatomic:* Nezaručuje bezpečný přístup více vláken k hodnotám. Používá se pro svoji rychlost. Není defaultně nastaven.

Property může být deklarováno ručně, nebo pomocí *interface builder* aplikace, která ho vytvoří snadněji (pouze k vizuálním objektům). Syntaxe deklarace *property* je následující:

- *Syntaxe:* `@property (atributy) datovýTyp *název;`
- *Příklad 1:* `@property (nonatomic, strong) NSMutableArray *poleKategorii; //ukazatel na NSMutableArray`

Pokud bychom požadovali přístup v jiném souboru, je potřebné kromě importování hlavičkového souboru původní třídy také vytvořit instanci dané třídy. Teprve potom lze k danému *property* přistupovat pomocí tečkové notace (Rypress, 2015).

- *Příklad 2:* `self.poleKategorii;`

6.3.4 Zprávy Objective-C

Základní vlastností Objective-C a objektových jazyků obecně je schopnost zasílat zprávy objektům. V tomto případě je daná vlastnost realizována pomocí syntaxe hranatých závorek převzaté z jazyka Smalltalk.

6.3.4.1 Anatomie zasílání zpráv

Každá zpráva musí obsahovat *příjemce* a *selector*. Příjemce je objekt, který zprávu obdrží a selector je metoda, která je volána. Kromě těchto dvou prvků může mít zpráva ještě n *parametrů*. Počet argumentů závisí na počtu selektorů (parametrů metody viz. později).

- *Syntaxe:* `[příjemce selektor: parametr];`
- *Příklad:*

```
NSMutableArray *pole; //vytvoří instanci od třídy NSMutableArray, pointer
ukazující na pozici v paměti
[pole objectAtIndex: 0]; //pole je příjemce zprávy objectAtIndex, která
zjistí obsah na indexu 0
```

6.3.4.2 Zasílání vnořených zpráv (nesting)

Vlastnost zjednodušující psaní zdrojového kódu. Jedná se o zasílání více zpráv v jednom příkazu. Nejdřív se zašle zpráva objektu nejvíce vnořená. V příkladu je tedy nejdříve obstarána komunikace `[NSDate alloc]`. `Alloc` vrací pointer na nově vytvořenou instanci, kterou je potřeba inicializovat. Tato instance již existuje v paměti, ale je nutné mu zaslat metodu `init`, která ji připraví na přijímání dalších zpráv (Hillegass a kol., 2013).

- *Příklad:* `NSDate *dnesniDatum = [[NSDate alloc]init];`

6.3.5 Metody Objective-C

Jak již bylo zmíněno dříve, metody reprezentují chování objektu. Přesněji obsahují kód, který má být přeložen na základě zaslané zprávy danému objektu. Metody jsou v Objective-C deklarovány v hlavičkové souboru, ale mohou být deklarovány také přímo v implementačním souboru za značkou `@implementation`. Přesněji v hlavičkovém souboru jsou deklarovány zprávy, na které umí třída a její instance reagovat. Pro zpracování zpráv se ale v Objective-C využívají stejnojmenné metody (Demčák, 2010).

6.3.5.1 Pojmenování metody

Panuje konvence úplného pojmenování metod a jejich parametrů. V důsledku je název se všemi parametry metody dlouhý, ale přesně čitelný.

- *Správný příklad:*

```
-(void)stahováníInformacíURL:(NSSring *)httpZdrojString
druhStazeneInformace:(NSString *)druhInformaceString;
```

- *Špatný příklad:*

```
-(void)infomaceURL:(NSSring *)zdroj info:(NSString *)druh;
```

Parametry metody by měly mít již při jejím tvoření definován název odpovídající požadované hodnotě tak, aby nedošlo k jakémukoliv omylu.

6.3.5.2 Anatomie metody (zprávy)

Při tvorbě metody je určeno několik vlastností. Rozeberme ukázku z předešlé podkapitoly.

```
-(void)stahovaniInformaciURL:(NSSring *)httpZdrojString
druhStazeneInformace:(NSString *)druhInformaceString;
```

- *Znaménko před metodou:* Znaménka „+“ nebo „-“ určují, zda je metoda *instanční* nebo *třídní*. Metody, které jsou definované jako třídní, budou fungovat jen pro danou třídu a ne její instance. Nejčastěji se používají pro tvorbu inicializačních metod. Instanční naopak lze volat na instanci třídy, ve které jsou deklarovány.
    - Příklad volání třídní metody:

        - `[Trida tridniMetoda];`
    - Příklad volání instanční metody:
        - `Trida *instanceTridy = [[Trida alloc]init];`
          `[instanceTridy instancniMetoda];`

- *Návratová hodnota:* určuje datový typ, který funkce vrací. V tomto případě *void.*
- *Název parametrů a jejich návratový datový typ:* v tomto případě *`stahovaniInformaciURL:(NSString *)`* a *`druhStazeneInformace(NSString *)`* definují jaké datové typy vstupů funkce očekává.
- *Vkládající argument: httpZdrojString* a *druhInformaceString* napomáhají v čitelnosti zdrojového kódu tak, že napovídají jaká informace je na daném místě očekávána.

6.3.5.3 Implementace Metody

Vše co bylo deklarováno v hlavičkovém souboru `.h` je poté implementováno v souboru `.m` a jinak tomu není ani u metod. Ty jsou implementovány následovně (Hulet, Mcqueen, 2015):

```
-(void)stahovaniInformaciURL:(NSSring *)httpZdrojString
druhStazeneInformace:(NSString *)druhInformaceString
{//kód}
```

6.4 Shrnutí kapitoly

V kapitole byl představen jazyk Objective-C počínaje jeho historií. Následně byly popsány základní vlastnosti a práce s jazykem jako takovým. Popis těchto vlastností je založen na teoretické znalosti problematiky objektového programování popsaného v kapitole 3. Tyto znalosti jsou nezbytné pro jakoukoliv tvorbu aplikací v daném jazyce.

7 Swift

Nový programovací jazyk Swift byl představen společností Apple Inc. na vývojářské konferenci WWDC 2014. Tento jazyk má doplňovat (nahrazovat) starší Objective-C a usnadňovat tvorbu nových aplikací. V kapitole jsou popsány hlavní přednosti a vlastnosti jazyka. Dále základní syntaxe a porovnání se starším Objective-C.

7.1 O Swiftu

Swift je nový programovací jazyk pro tvorbu aplikací operačních systémů iOS, OS X, tvOS a watchOS. Velice podstatnou vlastností je kompatibilita s Objective-C třídami. Tato kompatibilita je zajištěna LLVM kompilátorem.

7.1.1 Vznik Swiftu

S vývojem jazyka Swift započal Chris Lattner v červenci roku 2010. Později v roce 2011 poskytlo spolupráci řada dalších programátorů. V červenci roku 2013 se stal jazyk Swift primárním zaměřením skupiny *Apple Developer Tools* (Lattner, 2015).

7.1.2 Playground

Jednou z novinek Swiftu a zároveň prostředí Xcode je nový nástroj zvaný *Playground*. Zde je možné psát zdrojový kód, který se okamžitě překládá a je okamžitě zřejmý výstup i hodnoty v proměnných. Je určen především pro účely spojené s výukou a testováním jednoduchých částí kódu. *Playground* má vlastní typ souboru s koncovkou `.playground`.

Obrázek 8 - Playground

7.1.3 Pouze jediný typ souboru .swift

Již při tvorbě nového projektu je zřejmá jedna ze základních změn oproti Objective-C. Pro psaní zdrojového kódu je používán pouze jeden základní typ souboru `.swift`. Tyto soubory splňují funkci hlavičkového i implementačního souboru. To znamená, že metody, *property* a další objekty jsou deklarovány a implementovány zde.

- *Syntaxe:*

```
Import UIKit
Class Trida: NadTrida, Protokoly {
//property
//metody}
```

Tento systém práce v jednom souboru věci částečně zjednodušuje, jelikož se metody a *property* nemusí definovat zvlášť.

7.2 Základní syntaxe

Při práci s jazykem Swift je zapotřebí znát základní informace o proměnných, *property*, metodách a zprávách. Těmito problémy se zabývá následující kapitola.

7.2.1 Deklarování proměnných a konstant

Viditelným zjednodušením jsou možnosti deklarování nových proměnných a konstant. Proměnné používají klíčové slovo `var` a konstanty `let`. Pokud přiřazená hodnota je přesně určující datový typ není zapotřebí datový typ deklarovat.

- *Příklad 1:*

```
var čísloInt = 20 //hodnota je automaticky typu Int
var str = „Jsem typu String“
```

Pokud ovšem není hodnota zřejmá nebo není žádná, je nutné deklarovat proměnné explicitně dle příkladu 2. Zde je deklarováno *cisloDouble* explicitně. Kdyby bylo deklarováno stejně jako v Příkladu 1, pak by datový typ proměnné byl `Int`.

- *Příklad 2:* `var cisloDouble: Double = 20`

7.2.2 Optional chaining

Jednou z novinek představených v jazyce Swift je možnost nastavení volitelných hodnot u *property* nebo proměnných. Pokud je proměnná typu volitelná, může tedy nabývat hodnoty a nebo nenabývá vůbec žádné hodnoty. Značení pro volitelnou hodnoty je *„?“* a pro nucenou *„!“*.

- *Příklad 1:*

```
var cislo1 : int?
var cislo2 : int!
```

Dle příkladu *cislo1* může být `nil`, když se k němu přistupuje. Oproti tomu *cislo2* musí mít přiřazenou hodnotu vždy. Pokud by tomu tak nebylo, nastane chyba: *unexpectedly found nil while unwrapping an Optional value.* Tato chyba nám říká, že byla nalezena hodnota `nil` tam, kde by být neměla.

V detailnějším případě přistupování k *property* viz. příklad 2. V příkladu přiřazujeme do konstanty *str* typu `String` hodnotu textového pole, která musí mít *property text*, přičemž *property text* obsahuje hodnotu (Apple, 2015i).

- *Příklad 2:* `let str : String = TextovePole!.text!`

7.2.3 Tuples

Zajímavou novinkou je datový typ *Tuple*. Jedná se o datový typ, který může obsahovat různé datové typy. Základní operace by se daly přirovnat k chování typu *Array*. Vytvořit *tuple* se dá dvěma způsoby viz. příklad 1.

- *Příklad 1:*

```
let prvniTuple = („První“, 1)
let druhyTuple = (poradi: „První“, hodnota: 2) //pojmenované parametry
```

Přistupovat k hodnotám je možné několika způsoby, které záleží na situaci.

- *Příklad 2:*

```
let (poradiString, poradiInt) = prvniTuple //přiřadí hodnoty podle toho
jak jsou zapsány v původním tuple
let (_,poradiInt) = prvniTuple //přiřadí pouze číslo.
let poradiString = prvniTuple.0 //přiřadí hodnotu na první pozici,
přičemž pozice se v tuples se počítají od 0
 let poradiString = druhyTuple.poradi //pokud jsou hodnoty pojmenované je
možné přistupovat podle jména pomocí tečkové notace
```

Tuple může být použit i jako návratový typ metody (více o metodách později) (Codingexplorer, 2014b).

7.2.4 Property Swift

Jak již bylo řečeno, hlavním účelem *property* je vytvořit přístupové metody *getter* a *setter*. V jazyce Swift tomu není jinak. *Property* třídy je tedy přístupná třídě samotné a jejím instancím (defaultně jsou všechny *property* nastaveny jako *public*). *Property* mohou být nastaveny jako proměnné nebo konstanty v závislosti na potřebě.

- *Syntaxe:* `atributy var nazev : datovyTyp`
- *Příklad:* `weak var kategorie : NSArray`

Na příkladu je vidět rozdílná syntaxe. Označení `Property` chybí, atributy se nacházejí v úvodu. Tyto atributy mohou být pouze `weak a unknown`.

- *Weak*: Tvoří nevlastnický vztah, a tak nebrání systému ARC (viz. kapitola 6.2) (Apple, 2015b) zrušit referencovaný objekt. Používá se v případě existence možnosti ztráty hodnoty dané *property*.
- *Unknown:* Funguje stejně jako *weak* s rozdílem, že vždy obsahuje hodnotu.

K *property* se přistupuje standardně pomocí tečkové notace.

7.2.5 Metody Swift

Metody, jak již bylo uvedeno při vysvětlování objektového paradigmatu, upravují chování třídy. Metody jsou deklarovány a implementovány přímo ve třídě a souboru `.swift`. Primárně jsou veškeré metody nastaveny jako *internal* (Codingexplorer, 2014a).

- *Internal:* Metody jsou tedy přístupné své definované třídě a modulu svého zdroje. Tato vlastnost by se dala přirovnat k deklarování metody v hlavičkovém souboru.
- *Public:* Nastaví úplnou viditelnost.
- *Privat:* Povolí přístup pouze pro třídu, ve které je daná metoda deklarována.

7.2.5.1 Anatomie metody Swift

Metody by měly být při tvorbě pojmenovány co nepřesněji, stejně jako jejich parametry.

- *Syntaxe celé metody je následující:*

```
přístupovýAtribut func nazevMetody(parametr1: datovýTyp1, parametr2:
datovýTyp2) -> návratováHodnota{}
```

Jak již bylo řečeno, přístupový atribut je automaticky nastaven na *Internal.* Pokud není zadaná návratová hodnota je primárně nastavena jako *void.*

- *Příklad 1:*

```
private func sectiCisla(prvniCislo: Int, druheCislo: Int) -> Int {return
prvniCislo + druheCislo }
func nedelejNic () { //nedělá nic }
```

V příkladu je vidět první metoda s celkově rozepsanou syntaxí. Druhá metoda je nejjednodušší možná a nedělá nic.

V dřívější kapitole byl vysvětlován datový typ *tuples.* Jak byl zmíněno, tento typ může být použit i jako návratová hodnota metody dle následujícího příkladu. Zde je použit *tuple* s pojmenovanými parametry, nicméně použít se dá i zjednodušený zápis (`String,Int`).

- *Příkald 2:*

```
func vratTuple () -> (nazev: String, cislo: Int) { return (nazev, cislo)}
```

7.2.5.2 Přístup k metodám Swift

V jazyku Swift se zasílání zpráv a také vyvolání metody provádí pomocí tečkové notace. Použijeme-li příklad metody z minulé podkapitoly, můžeme metodu volat jako v následující ukázce (Apple, 2015h).

- *Syntaxe:* `příjemce.selektor(parametr)`
- *Příklad:* `self.nedelejNic()`

7.3 Hlavní rozdíly oproti Objective-C

Kapitola 6 zabývající se jazykem Objective-C poskytla základní znalosti o jazyku a práce s ním. Dle struktury kapitoly 6 byla vytvořena kapitola 7 popisující jazyk Swift. Již zde jsou patrné majoritní rozdíly mezi danými dvěma jazyky.

- Rozdíly:
    - Objective-C používá dva druhy souborů, oproti tomu Swift pouze jeden. Díky tomu je Swift na první pohled přehlednější (pokud jde o orientaci v souborech projektu).
    - Tvorba metody v Objective-C je syntakticky složitější než v jazyce Swift. Úplnost názvů metod a parametrů by měla být dodržována u obou jazyků.
    - Objective-C používá hranaté závorky pro zasílání zpráv objektům. Tento způsob je na první pohled jednoznačný, ale hůře přehledný oproti Swiftu, který používá tečkovou notaci. Díky tomu není na první pohled zcela zřejmé, zda se jedná o volání metody nebo o přístup k vlastnosti objektu (rozdílem jsou kulaté závorky a případně parametry metody).
    - Swift přichází s možností *volitelných hodnot.*
    - Nová datová struktura `tuples`, který může obsahovat hodnoty různých datových typů.
    - Při deklaraci proměnných nebo konstant v jazyku Swift, není nutné deklarovat datový typ pokud má přiřazenou přesnou inicializační hodnotu. Proměnné typované jako objekty se nadále musí inicializovat.
    - Dalším důležitým rozdílem, ačkoliv nebyl přímo zmíněn je absence *pointerů* (ukazatelů) v jazyce Swift, které nemusí být použity (ale mohou).
    - V jazyku Swift není nutné zakončovat příkazy středníkem.
    - Možnost generického programování (obecné programování).
    - Funkce jsou *first–class* (mohou být samy předány jako parametr nebo návratová hodnota jiné funkce) (James, 2014).

7.4 Shrnutí kapitoly

Na závěr teoretické části práce byl představen programovací jazyk Swift. V kapitole byla představena základní syntaxe a vlastnosti, jako podklad pro elementární znalost daného jazyka.

II. Vlastní práce

8 Návrh programu

V první fázi tvorby aplikace v praktické části je nutné se zabývat samotnou myšlenkou ukázkového programu, funkcionalitou a designem. Tyto vlastnosti jsou pro oba programy naprosto identické.

8.1 Vlastnosti

Nejobecnějším vyjádřením celkového pojetí a myšlenky aplikace je sousloví „Mobilní peněženka“. Tato peněženka je tedy zcela nezávislá na internetovém bankovnictví a tím pádem zápis změn položek probíhá manuálně ze strany uživatele.

8.1.1 Představa uplatnění aplikace

Na trhu se již pohybují aplikace tohoto druhu. Smyslem použití je udržovat si aktuální, nicméně přibližný přehled o pohybech finančních prostředků. Přibližný z toho důvodu, že uživatel musí sám na svoji zodpovědnost zaznamenávat své transakce manuálně. Pokud tak činí, může jednoduše sledovat své výdaje například za stravu nebo zábavu. Výhodou je tedy i zápis položek nakoupených / přijatých v hotovosti. Takové položky se bez podobného nástroje těžko sledují.

8.1.2 Hlavní funkční vlastnosti aplikace

Aplikace obsahuje následující základní funkcionality, které umožňují realizaci myšlenkového modelu fungování aplikace tak jak byl představen v předešlé části 8.1.1.

8.1.2.1 Přehled financí

Základní vlastností je nutnost vidět příjmy, výdaje a zůstatek, kterým daný uživatel disponuje. Tato vlastnost je prioritní, a proto je integrována na hlavní obrazovce aplikace ihned po spuštění.

8.1.2.2 Grafické vyjádření příjmů a výdajů

Další typickou funkcionalitou je způsob grafického vyjádření příjmů a výdajů v podobě grafu. Tento graf slouží na první pohled k intuitivnímu zhodnocení poměru vydaných a přijatých prostředků a měl by se nalézat na hlavní obrazovce pod číselným vyjádřením příjmů a výdajů.

8.1.2.3 Zadávání příjmů / výdajů za jednotlivé položky

Pro zápis příjmů a výdajů slouží separátní obrazovky. K těmto funkcím se lze dostat pomocí tlačítek na hlavní obrazovce. Při zadávání uživatel vkládá název položky, její cenu a následně vybírá z kategorie pod kterou položku přiřadí.

8.1.2.4 Seznam položek

Úplný seznam příjmových a výdajových položek je k nalezení na hlavní obrazovce pod tlačítkem Detail odkazující na danou obrazovku. Zde jsou jednoduchými barvami rozlišeny položky příjmů a výdajů.

8.1.2.5 Detail položek

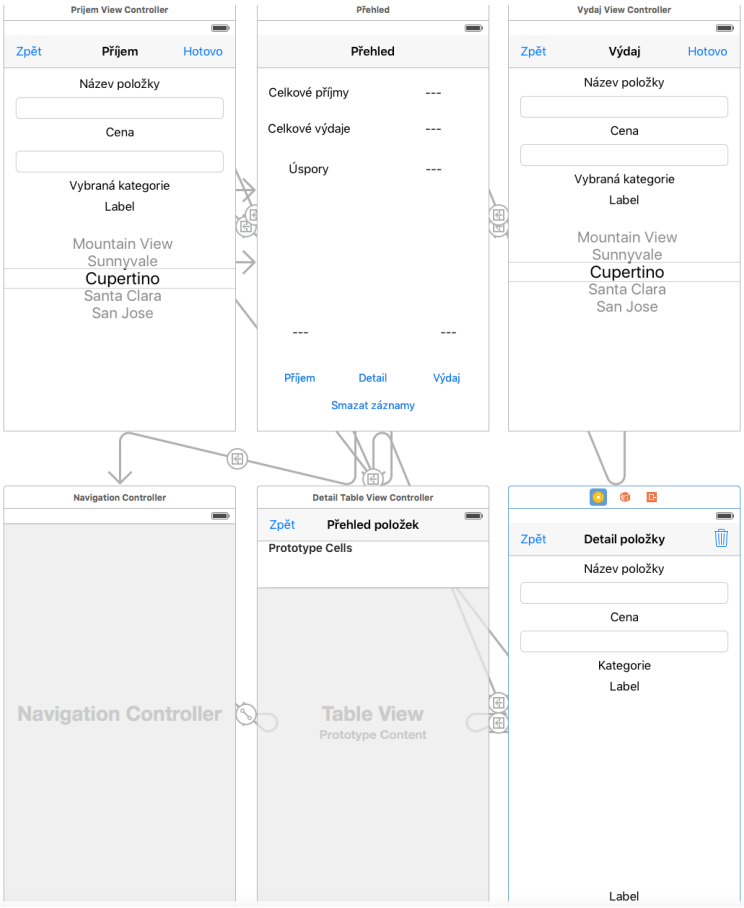
Do detailu dané položky je možné se dostat proklikem ze seznamu všech položek. V detailu jsou k nalezení informace o částce, názvu, kategorii a datu pořízení záznamu. Z tohoto detailu je rovněž možné položku smazat.

8.2 Design

Kapitola zabývající se rozvržením prvků na jednotlivých obrazovkách dle kapitoly 8.1.2.

8.2.1 Main storyboard

V první fázi tvorby aplikace po rozhodnutí, které funkce má aplikace obsahovat je nutné vytvořit a rozmístit prvky na jednotlivých obrazovkách. V této části není nutné psát žádný zdrojový kód, jelikož je používán *interface builder* (viz. kapitola Xcode). Prvky jsou tedy rozmístěny podle subjektivní představy. Pro každou obrazovku je následně vytvořen vlastní soubor (třída), který popisuje její chování.



Obrázek 9 - Main storyboard

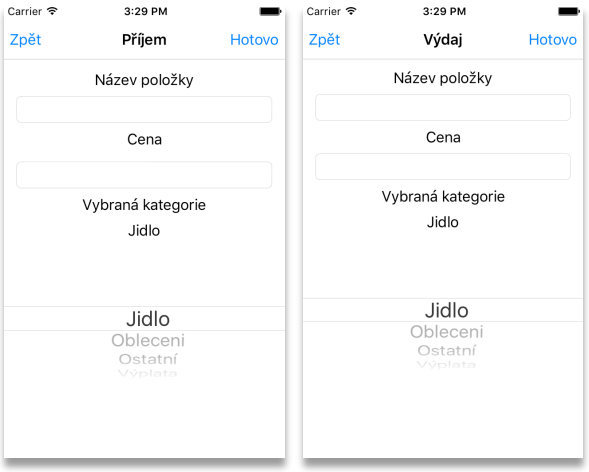
8.2.2 Hlavní obrazovka

Úkolem hlavní obrazovky je poskytnout nejpotřebnější informace na jednom místě a zároveň fungovat jako „rozcestník“ k funkcím dalším. K nalezení je zde přehled financí jak numericky, tak graficky ve formě poměru příjmů a výdajů. Ve spodní části jsou následně umístěná tlačítka pro přesměrování na obrazovky příjmů, výdajů a detailu položek.

Obrázek 10 - Hlavní obrazovka

8.2.3 Obrazovka příjem / výdaj

Jak je zřejmé z následujících obrázků, obrazovky příjmů a výdajů jsou téměř identické. Lze se k nim dostat pomocí tlačítek na hlavní obrazovce. Uživatel zde vyplňuje údaje o položce, kterou chce zaznamenat. Tyto informace se uloží a budou dostupné v detailu položky.

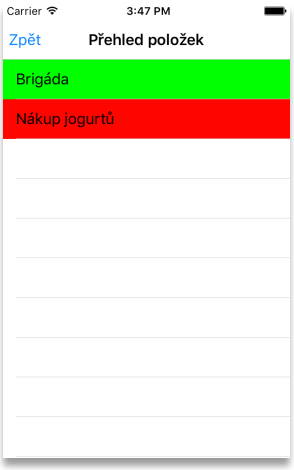


Obrázek 11 - Obrazovka příjem

Obrázek 12 - Obrazovka výdaj

8.2.4 Obrazovka seznamu položek

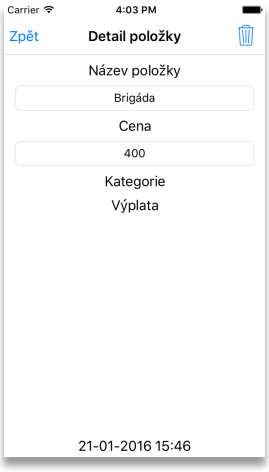
Do seznamu položek je možné se dostat skrze tlačítko „Detail“ na hlavní obrazovce. Zde nalezneme úplný přehled zaznamenaných položek, které jsou odděleny barvami podle toho zda je daná položka příjmem nebo výdajem. Po kliknutí na vybranou položku je uživateli zobrazen úplný detail.



Obrázek 13 - Obrazovka seznamu položek

8.2.5 Obrazovka detailu položky

V detailu položky jsou k nalezení veškeré informace, které uživatel zadal při jejím vytvoření. Ve spodní části je zobrazen datum a čas pořízení. V pravé horní části je potom možnost smazat danou položku ze záznamů (v případě chybného zadání například).



Obrázek 14 - Obrazovka detailu položky

8.3 Shrnutí kapitoly

V kapitole byla představena základní myšlenka a požadované vlastnosti tvořené aplikace. V závislosti na požadovaných vlastnostech bylo vytvořeno a demonstrováno adekvátní grafické rozhraní aplikace.

9 Dokumentace tvorby programu v Objective-C

Následující kapitola se zabývá tvorbou ukázkového programu v jazyce Objective-C. Zdrojový kód je rozebírán po jednotlivých souborech. Ukázky nepředstavují úplné znění zdrojového kódu aplikace.

9.1 HlavniViewController – Objective-C

V první podkapitole je popisována hlavní obrazovka, která slouží pro zobrazení elementárních informací podstatných pro uživatele ihned po spuštění aplikace.

9.1.1 Ukázka deklarace property, objektů uživatelského rozhraní, funkce

V Objective-C jsou deklarace prováděny v hlavičkovém souboru s koncovkou `.h`. Vlastnosti a funkce deklarované v hlavičkovém souboru mohou být použity i v jiných třídách po vytvoření příslušné instance (v tomto případě Outlet popisku).

```
@interface HlavniViewController : UIViewController
@property (strong, nonatomic) IBOutlet UILabel *PopisekPrijem;
- (IBAction)tlacitkoSmazat:(id)sender;
@end
```

Hlavičkový soubor je následně importován do implementačního souboru pomocí `#import HlavickovySubor.`

9.1.2 viewDidLoad

Funkce obsahující chování poté, co se okno načte. Jsou zde provedeny počáteční výpočty a nastavení.

- *Načtení příjmů a výdajů:* Načtení probíhá v obou případech stejně. Pomocí speciálního objektu se načtou data uložená v zařízení pod specifickým klíčovým slovem.

```
NSUserDefaults *defaultsPrijem = [NSUserDefaults standardUserDefaults];
NSString *pojmenovaniPrijem;
if ((pojmenovaniPrijem =[defaultsPrijem stringForKey:nacteniCenyPrijem]))
{
self.PopisekPrijem.text = pojmenovaniPrijem;
}
```

- *Výpočet úspor:* Hodnoty popisků příjmů a výdajů musí být převedeny do typu `Int`. Následně může být proveden jednoduchý přepočet.

```
int PopisekPrijemCislo = [self.PopisekPrijem.text intValue];
int PopisekVydajCislo = [self.PopisekVydaje.text intValue];
if ((PopisekPrijemCislo != 0) && (PopisekVydajCislo != 0)) {
int UsporyVypocet = PopisekPrijemCislo - PopisekVydajCislo;
NSString *str = [NSString stringWithFormat:@"%d",UsporyVypocet];
self.PopisekUspory.text = str;
```

9.2 PrijemViewController a VydajViewController – Objective-C

Popis obou souborů bude popsán dohromady, jelikož mají mezi sebou pouze minoritní rozdíly. Uvedený kód se vztahuje k souboru *VydajViewController*. Tyto třídy obsahují popis chování obrazovek pro příjem a výdaj.

9.2.1 viewDidLoad

Po načtení okna je opět nutné provézt základní nastavení nutné pro další operace. V tomto případě je nastaven typ klávesnice pro dané textové pole, vytvořeno pole hodnot kategorií a načtení některých uložených dat.

```
self.HodnotaPolozky.keyboardType = UIKeyboardTypeNumberPad;
kategorie = [[NSArray
alloc]initWithObjects:@"Jidlo",@"Obleceni",@"Ostatni",@"Vyplata", nil];
NSUserDefaults *defaultsVydaj = [NSUserDefaults standardUserDefaults];
NSString *pojmenovaniVydaj;
if ((pojmenovaniVydaj =[defaultsVydaj stringForKey:nacteniCenyVydaj])) {
prenosStringuVydaj = pojmenovaniVydaj;
}
```

9.2.2 TlacitkoVydaj

V akci tlačítka je popsáno ukládání hodnot zadaných uživatelem pod specifický klíč. Nutné je také, aby hodnoty byli zadány úplně. Úplnost zadání řeší následující podmínka.

```
if (([self.NazevPolozky.text  isEqual: @""]) || (
[self.HodnotaPolozky.text  isEqual: @""]))
```

- *Zjištění data a času:* Pomocí `NSDateFormatter` je nastaven formát záznamu, který je následně převeden do textové podoby z aktuálního data a času.

```
NSDateFormatter *dateFormatter=[[NSDateFormatter alloc] init];
[dateFormatter setDateFormat:@"dd-MM-yyyy HH:mm"];
NSString *datum =[dateFormatter stringFromDate:[NSDate date]];
```

- *Ukládání dat:* V tomto případě je nutné podotknout, že ukládání dat muselo být rozděleno na dvě části. Pro jedno klíčové slovo se ukládají pouze názvy položek a pro jiné klíčové slovo se pak ukládají všechny informace o položce včetně jména. Ukládání názvu položek zvlášť bylo zavedeno z důvodu problému přidělování jména buňky v `tableView` z objektu `NSDictionary` obsahující veškeré údaje.

Proto jsou jména položek uloženy jako pole. V jazyce Swift tento problém nenastává.

1) Uložení pouze jména položky. Zde dojde k iteraci (průchodu) polem načtených hodnot a následné uložení hodnoty nové.

```
polozkyVydaj = [userDefaults objectForKey:@"polozky"];
NSMutableArray *Vydaj = [[NSMutableArray alloc]init];
[Vydaj addObject:self.NazevPolozky.text];
if (polozkyVydaj !=nil) {
NSMutableArray *novePole = [[NSMutableArray alloc]init];
NSMutableArray *dict = [[NSMutableArray alloc]init];
for (dict in polozkyVydaj) {
[novePole addObject:dict];
}
[userDefaults removeObjectForKey:@"polozky"];
[novePole addObject:[Vydaj objectAtIndex:0]];
[detailDefaults setObject:novePole forKey:@"polozky"];
} else {
[userDefaults removeObjectForKey:@"polozky"];
NSMutableArray *novePole = [[NSMutableArray alloc]init];
[novePole addObject:[Vydaj objectAtIndex:0]];
[userDefaults setObject:novePole forKey:@"polozky"];
}
```

Ukládání všech dat probíhá do objektu `NSDictionary`. Následně se provede iterace skrze načtená data. Ta se přiřadí do dočasného pole. Do tohoto pole se přidají nová data v objektu `NSDictionary`. Následně se veškerá data pro klíčové slovo smažou a znovu se pod dané klíčové slovo uloží právě toto pole obsahující i nové hodnoty.

```
detailVydaj = [detailDefaults objectForKey:@"detailPolozky"];
NSMutableDictionary *dataVydaj = [[NSMutableDictionary alloc]init];
[dataVydaj setObject:self.NazevPolozky.text forKey:@"nazevPolozky"];
[dataVydaj setObject:self.HodnotaPolozky.text forKey:@"hodnotaPolozky"];
[dataVydaj setObject:@"vydaj" forKey:@"druhZaznamu"];
[dataVydaj setObject:self.kategoriePopisek.text
forKey:@"kategoriePolozky"];
[dataVydaj setObject:datum forKey:@"datum"];
if (detailVydaj !=nil) {
NSMutableArray *novePole = [[NSMutableArray alloc]init];
NSDictionary *dict = [[NSDictionary alloc]init];
for (dict in detailVydaj) {
[novePole addObject:dict];
}
[detailDefaults removeObjectForKey:@"detailPolozky"];
[novePole addObject:dataVydaj];
[detailDefaults setObject:novePole forKey:@"detailPolozky"];
} else {
[detailDefaults removeObjectForKey:@"detailPolozky"];
NSMutableArray *novePole = [[NSMutableArray alloc]init];
[novePole addObject:dataVydaj];
[detailDefaults setObject:novePole forKey:@"detailPolozky"];
}
```

9.3 Graf – Objective-C

Následující podtřída `UIView` obsahuje vykreslení grafu, který vyjadřuje poměr příjmů a výdajů na hlavní obrazovce. Veškeré chování je implementováno ve funkci `drawRect:(CGRect)rect.`

- Nejprve proběhne načtení hodnot a jejich převedení do adekvátního tvaru, který může být následně použit.

```
NSUserDefaults *defaultsVydaj = [NSUserDefaults standardUserDefaults];
NSString *vydajString = [defaultsVydaj objectForKey:@"nacteniCenyVydaj"];
float vydajFloat = [vydajString floatValue];
```

- Následuje samotné vykreslení grafu, který má pevně danou pozici.

```
CGContextRef contextRef = UIGraphicsGetCurrentContext();
CGContextSetLineWidth(contextRef, 2.0);
CGContextSetRGBFillColor(contextRef, 0, 0, 1.0, 1.0);
CGContextSetRGBStrokeColor(contextRef, 0, 0, 1.0, 1.0);
CGColorRef red = [[UIColor redColor]CGColor];
CGColorRef green = [[UIColor greenColor]CGColor];
CGContextSetFillColorWithColor(contextRef, green);
CGRect PrijemPoint = (CGRectMake(80,200.0, 50.0, (-
prijemFloat/procento)*2));
CGContextFillRect(contextRef,PrijemPoint);
CGContextSetFillColorWithColor(contextRef, red);
CGRect VydajPoint = (CGRectMake(150, 200.0, 50.0, (-
vydajFloat/procento)*2));
CGContextFillRect(contextRef,VydajPoint);
```

9.4 DetailTableViewController – Objective-C

Hlavním úkolem této třídy a potažmo celé obrazovky je zobrazení seznamu položek a následné umožnění přechodu na detail položky.

9.4.1 Výpis položek z uložených hodnot

Výpis je proveden pomocí funkce, ve které se pracuje s buňkami. Tyto buňky mají identifikátory pro opakované použití. Do popisku buňky je pak přiřazen název položky z uložených dat. Tyto názvy jsou uloženy v samostatném poli jak již bylo řečeno.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"Cell" forIndexPath:indexPath];
static NSString *identifier = @"Cell";
if (cell == nil)
{
cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:identifier];
}
NSMutableArray *array = [self.polozkyView objectAtIndex:indexPath.row];
NSString *nazev = [NSString stringWithFormat:@"%@",array];
cell.textLabel.text = nazev;
```

9.4.2 Příprava na přenos dat mezi obrazovkami

Následující funkce identifikuje přechod mezi obrazovkami podle jedinečného identifikátoru. Poté je zjištěn index buňky, se kterou uživatel interaguje. Tento index bylo nutné v tomto případě uložit a na základě jeho hodnoty byla vybrána položka z pole knihoven (viz. 9.5.).

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
if ([segue.identifier  isEqual: @"Detail"]) {
cesta = [[NSIndexPath alloc]init];
cesta = self.tableView.indexPathForSelectedRow;
DetailViewController *prechodDetail = [segue destinationViewController];
prechodDetail.dataDetail = [polozkyView objectAtIndex:cesta.row];
NSUserDefaults *prechodDefaults = [NSUserDefaults standardUserDefaults];
[prechodDefaults setInteger:cesta.row forKey:@"index"];
}
```

9.5 DetailViewController – Objective-C

Implementuje chování obrazovky zobrazující detail vybrané položky.

9.5.1 viewDidLoad

Po zobrazení okna probíhá načtení všech dat a načtení indexu buňky. Dle indexu je vybrána pozice v poli načtených hodnot. Tato pozice obsahuje knihovnu s informacemi o vybrané položce. Poté je již možné přiřazovat jednotlivé hodnoty příslušným popiskům.

```
pomocnePole = [[defaultsDetail
objectForKey:@"detailPolozky"]mutableCopy];
NSInteger hodnotaIndexu = [indexDefaults integerForKey:@"index"];
self.dataDetail = [pomocnePole objectAtIndex:hodnotaIndexu];
self.NazevPolozkyDetail.text = [self.dataDetail
objectForKey:@"nazevPolozky"];
```

9.5.2 Tlačítko SmazatPoložku

Dvěma základními vlastnostmi je přepočet celkových příjmů a výdajů po odebrání položky a samotné odebrání položky.

- *Přepočet celkových příjmů a výdajů:* Je zjištěn druh položky, který se ukládá při jejím záznamu. Podlé toho je pak proveden výpočet a uložení hodnot.

```
if ([[self.dataDetail objectForKey:@"druhZaznamu"]  isEqual: @"prijem"] )
{
cenaPrijemCislo = cenaPrijemCislo - cenaPolozkyCislo;
NSString *str = [NSString stringWithFormat:@"%d", cenaPrijemCislo];
[defaultsPrijem setObject:str forKey:@"nacteniCeny"];
} else {
cenaVydajCislo = cenaVydajCislo - cenaPolozkyCislo;
NSString *str = [NSString stringWithFormat:@"%d", cenaVydajCislo];
[defaultsVydaj setObject:str forKey:@"nacteniCenyVydaj"];
}
```

- *Odstranění položek:* Probíhá nadvakrát jelikož je záznam uložen zvlášť pro název a zvlášť pro úplné informace položky. Samotné odstranění probíhá podobně jako ukládání. Odstraní se objekt v poli ležící na hodnotě indexu. Výsledné pole se opět uloží pod dané klíčové slovo.

```
NSMutableArray *pomocnePoleKnihoven = [[NSMutableArray alloc]init];
NSString *str = [[NSString alloc]init];
for (str in poleKnihovny) {
[pomocnePoleKnihoven addObject:str];
}
[pomocnePoleKnihoven removeObjectAtIndex:hodnotaIndexu];
[knihovnaDefaults removeObjectForKey:@"detailPolozky"];
[knihovnaDefaults setObject:pomocnePoleKnihoven forKey:@"detailPolozky"];
NSMutableArray *pomocnePolePolozek = [[NSMutableArray alloc]init];
NSString *strPolozek = [[NSString alloc]init];
for (strPolozek in polePolozek) {
[pomocnePolePolozek addObject:strPolozek];
}
[pomocnePolePolozek removeObjectAtIndex:hodnotaIndexu];
[userDefaults removeObjectForKey:@"polozky"];
[userDefaults setObject:pomocnePolePolozek forKey:@"polozky"]
```

9.6 Shrnutí kapitoly

V této kapitole byla popsána tvorba aplikace v programovacím jazyce Objective-C. Popis byl proveden po jednotlivých souborech (třídách) a příslušných metodách, které jsou pro fungování aplikace stěžejní.

10 Dokumentace tvorby programu ve Swift

V následující kapitole jsou popsány klíčové části kódu ukázkové aplikace v tomto případě v jazyce Swift. Vybrané ukázky kódu nepředstavují úplné znění celého zdrojového kódu aplikace. Zdrojový kód bude rozebírán po jednotlivých souborech aplikace a popisován.

10.1 HlavniViewController - Swift

Prvním popisovaným souborem bude popis chování *controlleru*, který řídí chování hlavní obrazovky. Tato obrazovka je načtena po spuštění aplikace jako první.

10.1.1 Deklarace proměnných a objektů v obrazovce

V horní části třídy jsou připojeny prvky z obrazovky jako jsou například tlačítka nebo popisky. Toto propojení se tvoří pomocí *interface builderu*, takže kód je generován.

```
@IBOutlet var PopisekVydajeProcenta: UILabel!
```

Dále je možné deklarovat třídní vlastnosti, ke kterým je potřeba přistupovat v celé třídě.

```
var nacteniCenyPrijem = "cenaPrijem"
```

Jak je vidět v tomto případě je potřeba k deklaraci proměnné pouze klíčově slovo `var` a následně přiřadit hodnotu. Datový typ se tak určí dle přiřazené hodnoty.

10.1.2 viewDidLoad

Funkce, ve které se provedou příkazy poté co je daná obrazovka načtena. Zde jsou zapsány potřebné počáteční nastavení a načtení dat.

- *Počáteční nastavení:* Nastaví barvy na tlačítkách a textu.

```
tlacitkoVydaj.backgroundColor = UIColor.redColor()
PopisekVydaje.textColor = UIColor.redColor()
```

- *Načtení příjmů a výdajů:* Načítání příjmů a výdajů je provedeno identickým postupem. Následující části kódu vytvoří objekt, který dokáže uchovávat data pod daným klíčem. Data se tedy přiřadí danému popisku. Ukládání dat pod klíče probíhá v souborech pro zadávání příjmů a výdajů.

```
let defaultsPrijem = NSUserDefaults.standardUserDefaults()
if let pojmenovaniPrijem = defaultsPrijem.stringForKey(nacteniCenyPrijem)
{
self.PopisekPrijem.text = pojmenovaniPrijem
}
```

- *Výpočet úspor:* Nejprve je nutné převézt textový obsah popisků do číselného typu. Následně je ošetřena podmínka tak, aby hodnoty nemohly být `nil`. Zde je poté proveden samotný přepočet hodnot.

```
let PopisekPrijemCislo = Int(PopisekPrijem.text!)
let PopisekVydajCislo = Int(PopisekVydaje.text!)
if ((PopisekPrijemCislo != nil) && (PopisekVydajCislo != nil))
{
let UsporyVypocet = PopisekPrijemCislo! - PopisekVydajCislo!
PopisekUspory.text = String(UsporyVypocet)
}
```

10.2 PrijemViewController a VydajViewController – Swift

Popis obou souborů bude proveden dohromady, jelikož mají mezi sebou pouze minoritní rozdíly. Uvedený kód se vztahuje k souboru *VydajViewController*. Tyto třídy obsahují popis chování obrazovek pro příjem a výdaj.

10.2.1 Deklarace proměnných a objektů v obrazovce

V následujících řádcích je vidět deklarace proměnných i s určitým datovým typem.

```
@IBOutlet var kategoriePopisek: UILabel!
var kategorie : NSArray!
var cisloPuvodniVydaj : Int?
```

10.2.2 viewDidLoad

Po načtení obrazovky je opět nutné provézt základní nastavení a načtení hodnot.

- *Počáteční nastavení:* Nutné nastavení typu klávesnice pro dané textové pole a naplnění pole kategorií.

```
self.HodnotaPolozky.keyboardType = UIKeyboardType.NumberPad
kategorie = ["Jidlo","Obleceni","Ostatní","Výplata"]
```

- *Načtení dat:* Načtení hodnot probíhá stejně jako je tomu na hlavní obrazovce.

10.2.3 TlačítkoVýdaj

Tímto tlačítkem se zadané hodnoty uloží pod daný klíč. Důležité je, aby byly hodnoty zadány, a proto je veškeré chování ošetřeno podmínkou, která zajišťuje úplnost vyplněných údajů.

```
if ((NazevPolozky.text?.isEmpty == true)
|| (HodnotaPolozky.text?.isEmpty == true))
```

Aby hodnota položek byla číselná, je zajištěno číselným typem klávesnice ve funkci `viewDidLoad()`.

- *Zjištění data a času:* Pomoci `NSDateFormatter` je nastaven formát záznamu, který je následně převeden do textové podoby z aktuálního data a času.

```
let DateFormatter = NSDateFormatter()
let date = NSDate()
DateFormatter.dateFormat = "dd-MM-YYYY HH:mm"
let datum = DateFormatter.stringFromDate(date)
```

- *Ukládání informací:*

1) Nejprve jsou již existující uložená data načtená do pole.

```
var polozkyVydaj: NSMutableArray? = userDefaults.objectForKey("polozka")
as? NSMutableArray
```

2) Následuje zápis zadaných informací do objektu typu `NSMutableDictionary`. Zde se data uloží pod specifický klíč.

```
dataVydaj.setObject(HodnotaPolozky!.text!,forKey: "hodnotaPolozky")
dataVydaj.setObject("vydaj",forKey: "druhZaznamu")
```

3) Jako poslední krok je nutné iterovat skrze již uložené informace. Tyto informace uložené v poli obsahující objekty knihoven se za sebe přiřadí do pomocného pole. Vše, co bylo uložené pod klíčovým slovem, se smaže a nahradí aktuálním polem knihoven, ve kterém je přiřazena i knihovna aktuálně zadaných informací. Dojde k uložení informací pod klíčové slovo. Pokud se jedná o první záznam k iteraci nedochází a provádí se záznam rovnou.

```
if ((polozkyVydaj) != nil) {
let novePole:NSMutableArray = NSMutableArray()
for dict:AnyObject in polozkyVydaj!{
novePole.addObject(dict as! NSDictionary)
}
userDefaults.removeObjectForKey("polozka")
novePole.addObject(dataVydaj)
userDefaults.setObject(novePole, forKey: "polozka")
```

```
}else{
userDefaults.removeObjectForKey("polozka")
polozkyVydaj = NSMutableArray()
polozkyVydaj!.addObject(dataVydaj)
userDefaults.setObject(polozkyVydaj, forKey: "polozka")
}
```

10.3 Graf – Swift

Následující část obsahuje zdrojový kód pro jednoduše kreslený graf vyjadřující poměr příjmů a výdajů. Graf se vykresluje na hlavní obrazovce, ale ve vlastní třídě `UIView`. Veškerý kód se nachází ve funkci `drawRect(rect: CGRect)`.

- Nejprve je nutné načíst hodnoty a převézt na správný tvar tak, aby mohly být použity při vykreslení.

```
if ((defaultsPrijem.objectForKey("cenaPrijem") != nil) &&
(defaultsVydaj.objectForKey("cenaVydaj") != nil))
{
let prijemString : NSString = defaultsPrijem.objectForKey("cenaPrijem")
as! NSString
let vydajString : NSString = defaultsVydaj.objectForKey("cenaVydaj") as!
NSString
let prijemFloat = prijemString.floatValue
let vydajFloat = vydajString.floatValue
let procento = CGFloat(((prijemFloat + vydajFloat)/100))
let prijemCGFloat : CGFloat = (-CGFloat(prijemFloat)/procento)*2
let vydajCGFloat : CGFloat = (-CGFloat(vydajFloat)/procento)*2
```

- Samotné vykreslení probíhá na pevně zadaných souřadnicích. Hodnoty jsou spočítány zvlášť, jelikož ve funkci `CGRectMake()` nelze pracovat se znamínky v jazyce Swift.

```
let contextRef = UIGraphicsGetCurrentContext()
CGContextSetLineWidth(contextRef, 2.0)
let red : CGColorRef = UIColor.redColor().CGColor
let green : CGColorRef = UIColor.greenColor().CGColor
let prijemGraf = CGRectMake(80, 200, 50, prijemCGFloat)
CGContextAddRect(contextRef, prijemGraf)
CGContextSetFillColorWithColor(contextRef, green)
CGContextFillRect(contextRef,prijemGraf)
let vydajGraf = CGRectMake(150, 200, 50, vydajCGFloat)
CGContextAddRect(contextRef, vydajGraf)
CGContextSetFillColorWithColor(contextRef, red)
CGContextFillRect(contextRef,vydajGraf)}
```

10.4 DetailTableViewController - Swift

Tato třída ovládající seznam položek je velice důležitou součástí aplikace. Nejdůležitější součástí je zobrazení uložených dat do seznamu a následné zajištění přesměrování správných dat do okna s detailem položky. Načtení položek probíhá stejným způsobem, jako tomu bylo v předešlých případech.

10.4.1 Výpis položek z uložených hodnot

Hlavní účel celé třídy. Výpis je proveden pomocí funkce ve které se pracuje s buňkami. Tyto buňky mají identifikátory pro opakované použití. Do popisku buňky je pak přiřazen název položky z uložených dat.

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell {
let cell = tableView.dequeueReusableCellWithIdentifier("Cell",
forIndexPath: indexPath)
let jednaPolozka:NSDictionary = polozkyView.objectAtIndex(indexPath.row)
as! NSDictionary
cell.textLabel?.text = jednaPolozka.objectForKey("nazevPolozky") as?
String
return cell
}
```

10.4.2 Příprava na přenos dat mezi obrazovkami

V následující funkci je identifikováno propojení mezi obrazovkami pomocí jedinečného identifikátoru. Následně je určená řádka v seznamu, která byla vybrána. Nakonec se data pro vybraný řádek přiřadí proměnné v `DetailPolozkyViewController.swift`

```
override func prepareForSegue(segue: UIStoryboardSegue?, sender:
AnyObject?) {
if (segue!.identifier == "Detail")
{
var selectedIndexPath: NSIndexPath!
selectedIndexPath = self.tableView.indexPathForSelectedRow
let prechodDetail = segue!.destinationViewController as!
DetailPolozkyViewController
prechodDetail.dataDetail =
polozkyView.objectAtIndex(selectedIndexPath.row) as! NSDictionary
}
      }
```

10.5 DetailPolozkyViewController – Swift

Třída implementující chování obrazovky detailu vybrané položky.

10.5.1 viewDidLoad

V tomto případě zde probíhá zablokování interakce uživatele s textovými poli a načítání hodnot pro danou položku.

```
HodnotaPolozkyDetail.userInteractionEnabled = false
kategorieDetailpopisek.text = dataDetail.objectForKey("kategoriePolozky")
as? String
```

10.5.2 Tlačítko pro smazání položky

Toto tlačítko má dvě důležité funkce. První je úprava hodnot celkových příjmů a výdajů na hlavní obrazovce, pokud je nějaká položka smazána. Druhou vlastností je samotné

vyřazení záznamu o dané položce. Deklarace všech proměnných v následujících ukázkách nejsou uvedeny.

- *Přepočet hodnot celkových příjmů nebo výdajů:* Podmínkou se ověří, zda jde o příjem nebo výdaj a následně se provede daný přepočet. Výsledek se poté opět uloží.

```
if ((self.dataDetail.objectForKey("druhZaznamu")!.isEqual("prijem"))){
cenaPrijemCislo = cenaPrijemCislo - cenaPolozkyCislo!
let str : NSString! = String(cenaPrijemCislo)
defaultsPrijem.setObject(str, forKey: "cenaPrijem")} else {
cenaVydajCislo = cenaVydajCislo - cenaPolozkyCislo!
let str : NSString! = String(cenaVydajCislo)
defaultsVydaj.setObject(str, forKey: "cenaVydaj")
}
```

- *Smazání záznamu:* Prvním krokem je načíst všechna data. Následně se provede iterace těmito daty a přidělí se do pomocného pole. Z tohoto pole se následně odebere položka, která je právě mazána. Záznam pro klíčové slovo se smaže a následně se pod dané klíčově slovo uloží pomocné pole bez mazané položky.

```
for knihovna:AnyObject in polePolozek{
promenePolePolozek.addObject(knihovna as! NSDictionary) }
promenePolePolozek.removeObject(dataDetail)
userDefaults.removeObjectForKey("polozka")
userDefaults.setObject(promenePolePolozek, forKey: "polozka")
```

10.6 Shrnutí kapitoly

V této kapitole byla popsána tvorba aplikace v programovacím jazyce Swift. Popis byl proveden po jednotlivých souborech (třídách) a příslušných metodách, které jsou pro fungování aplikace stěžejní.

11 Diskuze - Výsledné zhodnocení pozorování jazyků

Mnoho znalostí a vlastností popsaných v teoretické části práci bylo možné pozorovat v praktické části.

Tvorba kódu v jazyce Objective-C je na první pohled rozsáhlejší a složitější. Komplikovanější je například z důvodu potřeby extra souboru na deklarování, delších syntaktický struktur jako je zasílání zpráv, nebo deklarace metod (zpráv). Z těchto důvodů je zdrojový kód v Objective-C delší a obsáhlejší. Dále bylo nutné čelit problémům při přidělování určitého objektu jako zdroje pro plnění buněk v *tableView*. Díky tomu muselo dojít k tvorbě dalších částí kódu při ukládání informací a předávání informací v detailu objektu, jak bylo popsáno v praktické části práce.

Swift je oproti Objective-C velice intuitivní a jednoduchý na čtení. Absence práce s ukazateli a jednoduchá inicializace objektů je velice příjemná. Automatické přidělování základních datových typů proměnným a konstantám zjednodušuje a urychluje tvorbu kódu. Jazyk Swift byl představen s myšlenkou zjednodušení a zefektivnění, a to se rozhodně povedlo.

Oba jazyky používají stejnou sadu základních knihoven tříd a jejich metod. Jak bylo uvedeno, tak v syntaxi a některých postupech jsou provedeny změny. Nicméně jazyky jsou si dost podobné. Pokud tedy programátor plně ovládá jazyk Objective-C, musí pro něj být velice jednoduché a příjemné začít tvořit v novějším Swiftu. Díky těmto vlastnostem, což potvrzují moje zkušenosti, je přechod ze starého Objective-C na nový Swift velice hladký.

Výsledné pozorování a vyzkoušení obou jazyků mě tedy přesvědčilo o tom, že jazyk Swift je plnohodnotný a efektivní aplikační programovací jazyk. To musí mít za následek jeho častější použití a následně možná i plné nahrazení staršího Objective-C.

12 Závěr

V bakalářské práci jsem se zabýval programovacími jazyky Objective-C a Swift a jejich rozdíly, které jsem prakticky vyzkoušel a demonstroval v praktické části práce na ukázce malé aplikace.

V úvodní části teoretické poloviny práce jsem jednoduše nastínil teoretické podklady objektového programování spolu se základními pojmy. Ty jsou následně používány skrze celou práci. Dále jsem se okrajově věnoval představení jednotlivých verzí operačního systému iOS z důvodu poukázání na vývoj jeho vlastností a funkcí. Další kapitola pojednává o prostředí Xcode a jeho částech. Základní orientace v tomto prostředí je nutnou znalostí pro tvorbu aplikací pomocí Objective-C a Swift.

V druhé polovině teoretické části jsem se zabýval jednotlivými jazyky. Zde byly vysvětleny základní syntaxe jako je deklarování proměnných, tvorba metod, zasílání zpráv a další specifické vlastnosti jednotlivých jazyků. V závěru teoretické části byly vyzdviženy některé rozdíly jazyků.

Praktická část byla věnována ukázkovým aplikacím v obou jazycích. Popis byl proveden po jednotlivých souborech a v nich po jednotlivých funkcích či klíčových částech zdrojového kódu. Zmíněny byli i případné problémy při jejich tvorbě.

13 Seznam použitých zdrojů

Tištěné zdroje:

ČADA, Ondřej, 2009a *Cocoa: úvod do programování počítačů Apple*. 1. vyd. Praha: Grada. 199 s. ISBN 978-80-247-2778-3.

ČADA, Ondřej, 2009b. *Objektové programování: naučte se pravidla objektového myšlení*. 1. Vyd. Praha: Grada. 200 s. ISBN 978-80-247-2745-5.

HILLEGASS, A. WARD, M, 2013. *Objective-C Programming: The Big Nerd Ranch Guide*. Atlanta: Big Nerd Ranch Guides. 325 s. ISBN 978-0321942067.

KEUR, C. HILLEGASS, A. CONWAY, J, 2014. *iOS Programming: The Big Nerd Ranch Guide*. Atlanta: Big Nerd Ranch Guides. 560 s. ISBN 978-0321942050.

KOCHAN, Stephen G, 2010. *Objective-C 2.0: výukový kurz programování pro Mac OS X a iPhone.* Vyd. 1. Brno: Computer Press. 550 s. ISBN 978-80-251-2654-7.

MERUNKA, Vojtěch, 2008. *Objektové modelování.* 1. vyd. Praha: Alfa Nakladatelství. 197 s. ISBN 978-80-87197-04-2.

Elektronické zdroje:

ALZA, 2015a. *Tak šel čas s iOS, část 1*. [online]. [cit. 2015-09-05]. Dostupné z: https://m.alza.cz/tak-sel-cas-s-ios-cast-1-art10970.htm.

ALZA, 2015b. *Tak šel čas s iOS, část 2*. [online]. [cit. 2015-09-07]. Dostupné z: https://www.alza.cz/tak-sel-cas-s-ios-cast-2-art10980.htm.

APPLE INC, 2015a. *About the iOS Technologies*. [online]. [cit. 2015-09-10]. Dostupné z: https://goo.gl/iG0TTQ.

APPLE INC, 2015b. *Automatic Reference Counting*. [online]. [cit. 2015-10-07]. Dostupné z: https://goo.gl/0MOtPP.

APPLE INC, 2015c. *Cocoa Touch Layer*. [online]. [cit. 2015-09-10]. Dostupné z: https://goo.gl/mDFIpR.

APPLE INC, 2015d. *Core OS Layer*. [online]. [cit. 2015-09-10]. Dostupné z: https://goo.gl/D2xpn3.

APPLE INC, 2015e. *Core Service Layer*. [online]. [cit. 2015-09-10]. Dostupné z: https://goo.gl/aARrjq.

APPLE INC, 2015f. *LLVM Compiler Overview*. [online]. [cit. 2015-09-11]. Dostupné z: https://goo.gl/nO2spZ.

APPLE INC, 2015g. *Media Layer*. [online]. [cit. 2015-09-10]. Dostupné z: https://goo.gl/M2RkWb.

APPLE INC, 2015h. *Methods*. [online]. [cit. 2015-10-07]. Dostupné z: https://goo.gl/UQe236.

APPLE INC, 2015i. *Optional Chaining*. [online]. [cit. 2015-09-20]. Dostupné z: https://goo.gl/e6Wloa.

APPLEDYSTOPIA, 2015. *iOS 9 Features*. [online]. [cit. 2015-09-07]. Dostupné z: http://www.appledystopia.com/news/ios-9-features-page-1/.

BAYER, Tomáš, 2015. *Úvod do OOP*. [online]. [cit. 2015-09-03]. Dostupné z: https://web.natur.cuni.cz/~bayertom/Prog2/prog2_1.pdf.

BUSTER, Hein, 2013. *The Evolution Of iOS: From iPhone OS To iOS 7*. [online]. 18. 8. 2013 [cit. 2015-09-05]. Dostupné z: http://www.cultofmac.com/191340/the-evolution-of-ios-from-iphone-os-to-ios-6-gallery/.

CODINGEXPLORER, 2014a. *Access Control in Swift*. [online]. 23.7.2014 [cit. 2015-10-07]. Dostupné z: http://www.codingexplorer.com/access-control-swift/.

CODINGEXPLORER, 2014b. *Tuples in Swift: Create, Read, and Return*. [online]. 6.10.2014 [cit. 2015-09-20]. Dostupné z: http://www.codingexplorer.com/tuples-in-swift-create-read-and-return/.

ČIŽMÁR, Michal, 2005. *Programujeme v jazyku C# - diel 32. – OOP V*. [online]. 24. 8. 2005 [cit. 2015-09-03]. Dostupné z: http://archiv.inet.sk/2425-programujeme-v-jazyku-c---diel-32---oop-v.html.

DEMČÁK, Marek, 2010. *Programovací jazyk Objective-C a učelnost jeho zařazení do výuky*. Praha: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií, 2010. Vedoucí diplomové práce: Ing. Rudolf Pecinovský, CSc. [cit. 2015-09-15]. Dostupné z: http://goo.gl/UDWQwP.

FOJTÍK, Rostislav, 2002. *Vývoj objektových aplikací 1*. [online]. 2002 [cit. 2015-09-03]. Dostupné z: http://www1.osu.cz/~fojtik/doc/VOA1.pdf.

FROST, James, 2015. *Instruments Tutorial with Swift: Getting Started*. [online]. 29. 5. 2015 [cit. 2015-09-15]. Dostupné z: http://www.raywenderlich.com/97886/instruments-tutorial-with-swift-getting-started.

HULET, Michael, MCQUEEN, Jacob, 2015. *In Objective-C, whats the difference between Instance and Class methods, and how do you switch between the two?*. [online].

[cit. 2015-09-20]. Dostupné z: https://teamtreehouse.com/community/in-objective-c-whats-the-difference-between-instance-and-class-methods-and-how-do-you-switch-between-the-two.

IMORE, 2015. *iOS 6*. [online]. [cit. 2015-09-07]. Dostupné z: http://www.imore.com/ios-6 .

IMORE, 2015. *iOS 8*. [online]. [cit. 2015-09-07]. Dostupné z: http://www.imore.com/ios-8.

IMORE, 2015. *iOS 9*. [online]. [cit. 2015-09-07]. Dostupné z: http://www.imore.com/ios-9.

JAMES, Mike, 2014. *Apple's New Language - Swift* [online]. 3.6.2014 [cit. 2015-10-07]. Dostupné z: http://www.i-programmer.info/news/201-ios/7378-apples-new-language-swift.html.

LATTNER, Chris, 2015. *Chris Lattner's Homepage*. [online]. [cit. 2015-09-20]. Dostupné z: http://nondot.org/sabre/.

LLVM, 2015. *LLVM Overview*. [online]. [cit. 2015-09-11]. Dostupné z: http://llvm.org.

LLVM, 2015. *The LLDB Debugger*. [online]. [cit. 2015-09-11]. Dostupné z: http://lldb.llvm.org.

RITCHIE, Rene, 2010. *iOS 4 review*. [online]. 14. 6. 2010 [cit. 2015-09-05]. Dostupné z: http://www.imore.com/ios-4-review.

RITCHIE, Rene, 2009. *iPhone OS 3.0 review*. [online]. 17. 6. 2009 [cit. 2015-09-05]. Dostupné z: http://www.imore.com/iphone-os-30-review.

RYPRESS, 2015. *Properties*. [online]. [cit. 2015-09-15]. Dostupné z: http://rypress.com/tutorials/objective-c/properties.

SEBASTIAN, Anthony, 2013. *Downgrading from iOS 7 to iOS 6: Why Apple won't let you*. [online]. 26. 8. 2013 [cit. 2015-09-07]. Dostupné z: http://www.extremetech.com/computing/167450-downgrading-from-ios-7-to-ios-6-why-apple-wont-let-you.

SVATOŠ, Patrik, PULTZNER Martin, 2015. *iOS 9: první dojmy z nového systému*. [online]. 18. 6. 2015 [cit. 2015-09-07]. Dostupné z: http://mobilenet.cz/clanky/ios-9-prvni-dojmy-z-noveho-systemu--20471.