Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Programovací jazyk Kotlin

Petr Ilek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Petr Ilek

Informatika

Název práce

Programovací jazyk Kotlin

Název anglicky

Kotlin Programming Language

Cíle práce

Práce je zaměřena na problematiku vývoje mobilních aplikací s využitím programovacího jazyka Kotlin. Hlavním cílem práce je představení jazyka a demonstrace jeho možností prostřednictvím implementace ukázkové mobilní aplikace. Dále bude provedeno jeho porovnání s programovacím jazykem Java.

Metodika

Práce sestává ze dvou hlavních částí – přehledu teoretických východisek a praktické části.

Metodika zpracování teoretické části je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána problematika vývoje aplikací v jazyce Kotlin.

V praktické části práce provedena implementace mobilní aplikace v tomto jazyce. Zkušenosti z vývoje aplikace budou shrnuty a bude provedeno porovnání základních rozdílů implementace v Kotlinu a Javě.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Kotlin, Java, JVM, Android, programování, mobilní aplikace

Doporučené zdroje informací

JEMEROV, Dmitry a Svetlana ISAKOVA. Kotlin in Action, 1st Edition, Manning Publications Co., 2017, 360 s. ISBN 9781617293290.

Kotlin Programming Language – Reference. Czech Republic: Prague. JetBrains [online]. Dostupné z WWW <https://kotlinlang.org/docs/reference/>.

LEIVA, Antonio. Kotlin for Android Developers, 1st Edition, CreateSpace Independent Publishing Platform, 2016, 200 s. ISBN 1530075610.

SAMUEL, Stephen a Stegan BOCUTIU. Programming Kotlin, Packt Publishing Limited, 2017, 420 s. ISBN 9781787126367.

VASIĆ, Miloš. Fundamental Kotlin, 1st Edition, Miloš Vasić, 2016, 155 s. ISBN 9788692030703.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Programovací jazyk Kotlin" 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 použitých zdrojů 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 15. 3. 2018 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu bakalářské práce panu Ing. Jiřímu Brožkovi, Ph.D. za vstřícný přístup a věcné připomínky. Dále bych chtěl také poděkovat všem blízkým, kteří mě podporovali v průběhu práce.

Programovací jazyk Kotlin

Abstrakt

Tato práce se zabývá programovacím jazykem Kotlin, který byl v roce 2017 představen jako oficiálně podporovaný programovací jazyk pro operační systém Android. Cílem práce bylo představení jazyka a demonstrace jeho možností prostřednictvím implementace ukázkové mobilní aplikace. Očekávalo se zároveň porovnání programovacích jazyků Kotlin a Java. Teoretická část byla založena na studiu odborných informačních zdrojů. Poznatky z teoretické části byly poté využity v praktické části práce. Výsledkem je funkční mobilní aplikace pro operační systém Android, která umožňuje studentům univerzity efektivní práci s katalogem knih univerzitní knihovny v mobilním telefonu. Na základě poznatků získaných z implementace ukázkové aplikace bylo popsáno, jak lze řešit nejčastější problémy při vývoji mobilních aplikací pro Android pomocí Kotlinu. Řešení těchto problémů bylo následně porovnáno s programovacím jazykem Java 8. Analýzou zdrojových kódů došlo k závěru, že použitím jazyku Kotlin lze dosáhnout stejné funkcionality pomocí menšího množství zdrojového kódu, než by tomu bylo v jazyce Java. To může vést k lepší čitelnosti a udržitelnosti kódu. Navíc, díky vzájemné interoperabilitě Kotlinu s Javou lze Kotlin používat v již odstartovaných projektech v Javě.

Klíčová slova: Kotlin, Java, JVM, Android, programování, mobilní aplikace

Kotlin programming language

Abstract

This thesis focuses on Kotlin programming language, which was announced as an official programming language for Android development at Google I/O 2017. Main goal of this thesis was introduction of language and demonstration of its options by implementing sample mobile application. Next goal was compare Kotlin and Java programming languages. Theoretical part was based on studying technical information sources. Knowledge from theoretical part was applied in practical part. In practical part was implemented sample mobile application in Kotlin. The result is working Android mobile application for university library catalogue. With experience from implementation was described how to solve common problems with Kotlin during development of Android mobile applications. Solutions of this problems were compared with Java 8 programming language. This comparison showed that what is possible to do in Java can be done in Kotlin with less lines of code which can lead to better maintainability of code. Thanks to interoperability of Kotlin with Java it is possible to use Kotlin in already started projects in Java.

Keywords: Kotlin, Java, JVM, Android, programming, mobile app

Obsah

Seznam obrázků

1 Úvod

Oficiálním programovacím jazykem pro vývoj nativních aplikací pro platformu Android byla donedávna pouze Java. Na každoroční vývojářské konferenci pořádané společností Google s názvem Google I/O byl v roce 2017 představen druhý oficiálně podporovaný programovací jazyk pro Android – Kotlin. Tento jazyk vyvíjí společnost JetBrains, která je známá pro vytváření nástrojů zaměřených na tvorbu softwaru. Kotlin vznikl za účelem nabídnout vývojářům průmyslově spolehlivý objektově orientovaný jazyk, který je v mnoha ohledech efektivnější než Java, ale zároveň je s ní naprosto interoperabilní.

Tato práce je zaměřena na představení programovacího jazyka Kotlin 1.1 a demonstrace jeho možností prostřednictvím implementace ukázkové mobilní aplikace. Tato ukázková aplikace zjednodušuje studentům univerzity práci na mobilním telefonu s katalogem knihovny na Studijním a informačním centru. Dále práce obsahuje shrnutí zkušeností z vývoje aplikace a porovnání základních rozdílů implementace v Kotlinu a Javě.

2 Cíle práce a metodika

2.1 Cíle práce

Práce je zaměřena na problematiku vývoje mobilních aplikací s využitím programovacího jazyka Kotlin. Hlavním cílem práce je představení jazyka a demonstrace jeho možností prostřednictvím implementace ukázkové mobilní aplikace. Dále bude provedeno porovnání s programovacím jazykem Java.

2.2 Metodika

Práce se skládá ze dvou hlavních částí – přehledu teoretických východisek a praktické části.

Metodika zpracování teoretické části je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána problematika vývoje aplikací v jazyce Kotlin.

V praktické části práce bude provedena implementace mobilní aplikace v tomto jazyce. Zkušenosti z vývoje aplikace budou shrnuty a bude provedeno porovnání základních rozdílů implementace v Kotlinu a Javě.

3 Teoretická východiska

3.1 Java

Java je objektově orientovaný programovací jazyk a platforma původně vyvíjená firmou Sun Microsystems v roce 1995. Jde o jeden z nejpopulárnějších programovacích jazyků na světě. Hlavním záměrem Javy je umožnit vývojářům napsat kód jednou a spustit ho všude, což znamená, že zkompilovaný Java kód může být spuštěn na všech platformách, které Javu podporují bez nutnosti rekompilace [1].

Zdrojové soubory jsou zkompilovány do formátu nazývaného bytekód, který spouští Java interpreter a runtime prostředí známé jako Java Virtual Machine.

V roce 2007 byla většina částí Javy uvolněna pod licencí GNU – General Public License. Roku 2009 byl Sun Microsystems koupen Oraclem a převzal tak dva klíčové produkty Javu a Solaris [2].

Při představení Javy ve verzi 8 došlo k mnoha novinkám. Mezi ně patří například možnost používat principy z funkcionálního programování a paralelní zpracování využívající streamy [2].

3.2 Java Virtual Machine

Java Virtual Machine neboli zkráceně JVM je softwarově založený stroj (tzv. virtuální stroj) od firmy Oracle, který umožňuje spustit téměř na každém zařízení (PC, Mac, mobil atd.) program napsaný v Javě nebo jiném jazyku, který je zkompilován do Java bytekódu. JVM obsahuje just-in-time (JIT) překladač, který překládá bytekód přímo do nativního strojového kódu počítače, na kterém je spuštěn, čímž dochází k urychlení jeho běhu. [3][4]

3.3 Programovací jazyk Kotlin

Kotlin je staticky typovaný programovací jazyk vyvíjen firmou JetBrains, která je známá tvorbou nástrojů pro vývoj softwaru. Projekt začal v roce 2010 a k prvnímu oficiálnímu vydání došlo v únoru 2016. Je vyvíjen pod licencí Apache 2.0 a jeho zdrojový kód je dostupný na internetu [5].

Kotlin je praktický jazyk založený na zkušenostech z průmyslu a jeho vlastnosti jsou určeny k pokrytí use casů mnoha softwarovými vývojáři. Vývojáři v JetBrains používali ranné verze Kotlinu několik let a na základě jejich odezvy došlo k vydání stabilní verze.

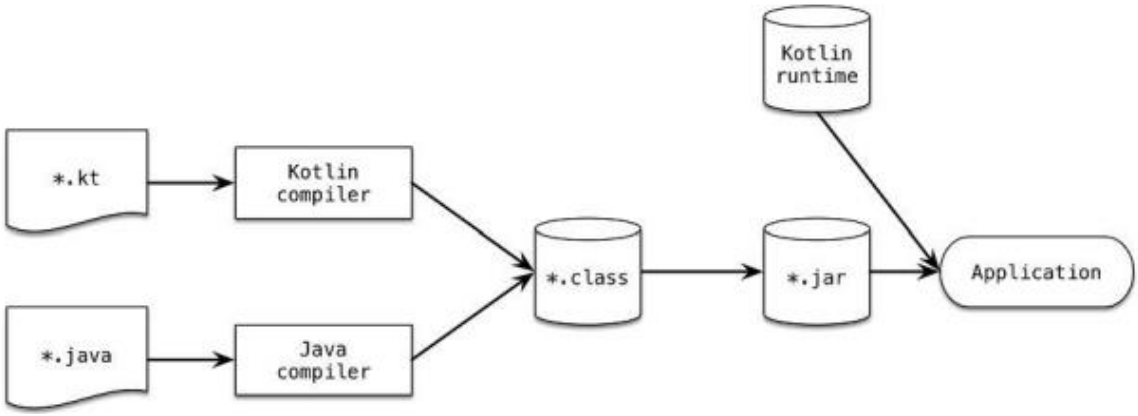
Kotlin se tedy spoléhá na vlastnosti a řešení, které se již objevily v jiných programovacích jazycích a ukázaly se jako úspěšné. Kotlin nenutí vývojáře k použití konkrétního programovacímu stylu nebo paradigmatu. Při jeho učení je možné používat techniky známé například z jazyku Java. [6].

Kotlin je možné použít na aplikace běžící na JVM, Androidu a nově (ve verzi 1.1) také v prohlížeči pomocí kompilace do Javascriptu. Je stoprocentně interoperabilní s dnes velmi rozšířeným programovacím jazykem Java. Oproti Javě je stručnější a podporuje tvorbu non-nullable typů.

Může být použit pro mnoho druhů vývoje software např. server-side, client-side web a Android. Pomocí Kotlin/Native, který je ve fázi vývoje bude možné tvořit aplikace pro vestavěné systémy/IoT, macOS a iOS. Nyní se Kotlin nejčastěji používá na mobilní a server-side aplikace. Na konferenci Google I/O v roce 2017 byl Kotlin oznámen jako oficiální programovací jazyk pro Android [5].

3.3.1 Kompilace Kotlinu pro Java Virtual Machine

Zdrojový kód jazyku je uložen v souborech s příponou kt. Kompilátor Kotlinu tyto soubory analyzuje a vygeneruje soubory s příponou class (stejně jako Java kompilátor). Vygenerované soubory jsou zabaleny a spuštěny standartní rutinou, která závisí na typu dané aplikace.



Obrázek 1: Zjednodušený proces překladu Kotlinu, zdroj: [6]

Kód kompilovaný překladačem je závislý na knihovně Kotlin runtime library, ta obsahuje definice standartní knihovny Kotlinu a rozšíření, které Kotlin přidává ke standartnímu Java API. Tato knihovna musí být distribuována společně s celou aplikací.

Dnes se v praxi ke kompilaci používají build systémy jako je Maven a Gradle. Kotlin je se všemi těmito systémy plně kompatibilní a tyto nástroje zajistí přidání Kotlin runtime library jako závislost aplikace. [6]

3.3.2 Vlastnosti Kotlinu

3.3.2.1 *Staticky typovaný programovací jazyk*

Stejně jako Java, Kotlin je staticky typovaný programovací jazyk. To znamená, že typ každého výrazu je znám při kompilaci. Díky tomu může kompilátor validovat metody a proměnné, které se programátor snaží používat. Kotlin, ale na rozdíl od Javy nevyžaduje specifikaci typu vždy, protože v mnoha případech může být typ proměnné určen z kontextu. V kódu níže je deklarována proměnná x, jelikož je inicializována číselnou hodnotou, kompilátor Kotlinu automaticky pozná, že typ bude Int. [6]

```
val x = 1
```

Mezi výhody staticky typovaných jazyků patří:

- **Výkon** – Volání metod je rychlejší, protože není potřeba zjišťovat při běhu programu metody, které mají být zavolány.
- **Spolehlivost** – Překladač verifikuje správnost programu, takže je zde menší riziko pádu při běhu.
- **Udržovatelnost** – Práce s neznámým kódem je jednodušší, jelikož programátor vidí, s jakým typem objektu pracuje.
- **Podpora nástrojů** – refaktoring kódu, napovídání a jiné vlastnosti, které umožnuje vývojové prostředí. [6]

3.3.2.2 *Prvky z funkcionálního programování*

Funkcionální programování si zakládá na matematickém přístupu k řešení problémů. Jeho paradigma je založeno na zápisu programu ve tvaru výrazu. Nejdůležitějšími složkami těchto výrazů jsou funkce a jejich aplikace na argumenty. V porovnání s imperativním programováním se vyznačuje čistějším a jednodušším kódem. [7]
Klíčovými koncepty funkcionálního programování jsou:

- ***First-class functions*** – Práce s funkcemi jako s hodnotami. Funkce lze ukládat do proměnných, předávat jako parametr, nebo je vrátit z jiných funkcí. Kód je tedy

stručný, jelikož práce s funkcemi jako s hodnotou dává programátorovi více abstrakce, což vede k odstranění duplicit v kódu.

- **Immutabilita (Neměnnost)** – Práce s neměnnými objekty, jejichž stav nemůže být změněn po jejich vytvoření. Díky neměnným datovým strukturám se vylučuje možnost modifikovat stejná data z více vláken a díky tomu je program tzv. *thread safe*.
- **Žádné vedlejší efekty** – Práce s tzv. *pure functions*, které vracejí stejný výsledek pro stejný vstup a neměnní stav jiných objektů. Z toho vyplývá, že program je snazší otestovat, jelikož logika programu je více izolovaná a není tak potřeba sestavovat v kódu prostředí, na kterém by byly testy závislé.

Všechny výše uvedené koncepty je možné v Kotlinu použít. Kotlin však funkcionální styl programování nevynucuje, a je v něm tedy možné pracovat i s *mutable* (měnnými) daty a psát funkce, které mají vedlejší účinky. [6]

3.3.2.3 *Stručnost*

Je obecně známo, že vývojáři stráví více času čtením již napsaného kódu než psaním nového. Programátor nejprve hledá konkrétní úsek kódu, který je potřeba upravit. K tomu je potřeba často přečíst a porozumět kódu, který psali kolegové nebo někdo, kdo na projektu už nepracuje. Teprve po pochopení souvisejícího kódu je možné provést potřebné modifikace. Vývojáři Kotllinu se snažili, aby jeho syntaxe popisovala význam. Spousta standartní syntaxe z Javy jako jsou *gettery*, *settery* a logika předávání parametrů v konstruktoru jsou v Kotlinu implicitní a nezanáší tak zdrojový kód. Dalším příkladem dlouhého kódu je například přístup k elementu uvnitř kolekce, Kotlin má bohatou standartní knihovnu, která vývojářům umožňuje nahradit tyto dlouhé a opakující se sekce kódu. [6]

3.3.2.4 *Null safe*

Jedním z největších problémů mnoha programovacích jazyků (včetně Javy) je přístup k vlastnostem přes proměnné, jejichž hodnota je *null*. Tato akce totiž způsobí vyhození výjimky, která pokud není ošetřena, způsobí pád celé aplikace. Kotlin se toto vyhazování výjimek snaží omezit tím, že při standartní deklaraci proměnné nemůže být její hodnota *null*. Aby proměnná mohla změnit svou hodnotu na *null*, musí se při deklaraci jejího typu uvést na konec typu otazník – viz dále. [8]

V kódu níže je přiřazen null do proměnné nenulového typu – kompilátor zahlásí chybu.

```
var a: String = "abc"
a = null
```

Otazník za typem String v kódu níže říká, že proměnná **b** může obsahovat null – je tedy *nullable*. Přiřazení nullu do proměnné **b** nezpůsobí chybu při kompilaci.

```
var b: String? = "abc"
b = null
```

Zpřístupnění length v proměnné **b** níže, vyhodnotí překladač jako nebezpečnou a zahlásí chybu.

```
val x = b.length
```

[8]

Pokud programátor potřebuje přistupovat k vlastnostem proměnné **b,** může to udělat následujícími způsoby:

1. **Explicitní kontrolou, zda proměnná není null**

```
val x = if (b != null) b.length else -1
// Kompilátor zachytí kontrolu na null – dovolí zpřístupnění
```

2. **Použitím tzv. safe call operátoru - ?**

```
val x = b?.length
// Pokud proměnná b není null -> přiřadí do x b.length
// Pokud proměnná b   je null -> přiřadí do x null
// Typem proměnné x bude Int? -> nullable Int
```

3. **Použitím tzv. elvis operátoru - ?:**

```
val x = b?.length ?: -1
// Pokud výraz nalevo od ?: není null přiřadí se do proměnné x výraz
// nalevo, jinak se přiřadí výraz napravo
// Typem proměnné x bude Int? – nullable Int
```

4. **Použitím tzv. not-null assertion operátoru - !!**

```
val x = b!!.length
// Operátor !! převádí hodnotu na non-null, pokud je hodnota null
// dojde k vyhození výjimky
```

[8]

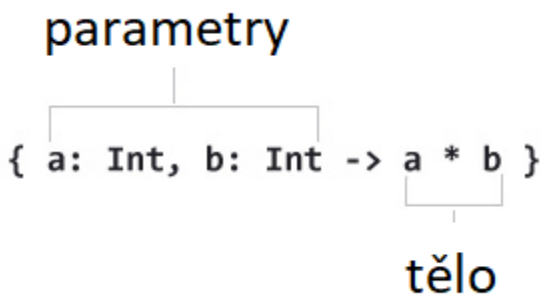
3.3.2.5 *Interoperabilita s Javou*

Kotlin je navržen tak, aby byl kompatibilní s existujícím Java kódem. Je z něj tedy možné volat Java knihovny a metody, dědit od Java tříd, implementovat Java rozhraní, používat Java anotace atd. Na rozdíl od jiných jazyků pro JVM, jde Kotlin s interoperabilitou ještě dále a umožňuje volat z Javy funkcionalitu napsanou v Kotlinu naprosto běžným způsobem. To vývojářům umožňuje přidat Kotlin do již existujících projektů. Kotlin nemá vlastní knihovnu kolekcí, jelikož využívá standartní knihovny Javy, případně od nich dědí a přidává funkcionalitu navíc. To znamená, že vývojáři nemusí převádět objekty z Javy do Kotlinu a naopak. [6]

3.3.3 **Přednosti Kotlinu**

3.3.3.1 *Lambda expression*

Lambda expression neboli lambda výraz je anonymní funkce – funkce beze jména. Tato funkce je předávána jako výraz bez deklarace. Na obr. 2 je zobrazena syntaxe deklarace lambda výrazu s parametry. Pokud lambda výraz nepřijímá žádné parametry, stačí místo deklarace parametrů napsat prázdné závorky.



Obrázek 2: Deklarace lambda výrazu, zdroj: [21]

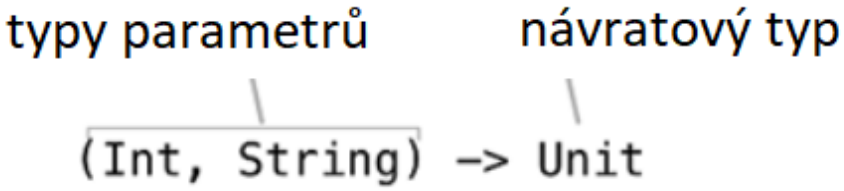
```
fun main(args: Array<String>) {
    val product = { a: Int, b: Int -> a * b }
    val result = product(9, 3)
}
```

Na příkladu výše je lambda výraz uložen do proměnné product. Tento lambda výraz přijímá dvě čísla typu Int a vrátí jejich součin. Do proměnné result se tedy uloží číslo 27. [21]

3.3.3.2 *High-Order Functions*

V Kotlinu je možné předávat funkci jako parametr do jiné funkce. Také je možné funkci z jiné funkce vracet. Tyto funkce se nazívají *High-Order Functions* neboli funkce vyššího řádu.

Pro funkci, která bude přijímat lambda výraz jako parametr je potřeba specifikovat funkci jako typ pro tento parametr – tzv. „function type“. Syntaxe je popsána na obr. 3 níže.



Obrázek 3: Deklarace funkce jako typu, zdroj [21]

Na příkladu níže je funkce vyššího řádu se jménem callMe, tato funkce přijímá funkci pod jménem greeting. Klíčové slovo Unit vyjadřuje, že funkce greeting nevrací žádnou hodnotu. Po spuštění programu dojde k vypsání „Hello!“ do konzole. [21]

```
fun callMe(greeting: () -> Unit) {
    greeting()
}

fun main(args: Array<String>) {
    callMe({ println("Hello!") })
}
```

Protože funkce callMe přijímá funkci jako jediný parametr, je možné ji zavolat kratším zápisem, ve kterém lze nahradit závorky pro parametry funkce samotným tělem funkce – viz níže.

```
fun callMe(greeting: () -> Unit) {
    greeting()
}

fun main(args: Array<String>) {
    callMe{ println("Hello!") }
}
```

3.3.3.3 *Data classes*

Kotlin představuje koncept datových tříd, které reprezentují jednoduché třídy sloužící jako kontejnery pro data – neobsahují logiku navíc. Tyto třídy se nazývají *data classes* a jsou označeny klíčovým slovem „data“. Struktura těchto tříd stačí definovat pouze pomocí konstruktoru. V parametrech konstruktoru se nadefinují atributy třídy a jejich měnitelnost/neměnitelnost. Viditelnost *getterů* a *setterů* je defaultně veřejná (public), pro jinou viditelnost je potřeba uvést ji před daný parametr konstruktoru. V kódu níže je vidět ukázka deklarace datové třídy User s atributy name a age. [18]

```
data class User(val name: String, val age: Int)
```

Kompilátor Kotlinu vygeneruje *gettery*, *settery* a implementaci:

- hashCode a equals – metody pro porovnání objektů
- toString – metoda pro výpis objektu ve formě „User(name=”Jan”, age=20)“
- copy – metoda pro vytvoření mělké kopie
- componentN – funkce pro tzv. „destructuring declarations“ viz dále

3.3.3.4 *Destructuring declarations*

„Destructuring declarations“ jsou deklarace umožňující jednoduchým způsobem extrahovat data z objektu do více proměnných pomocí deklarace níže.

```
val user = User("Jan", 20)
val (name, age) = user
```

V příkladu výše bude do proměnné name uložena hodnota „Jan“ a do proměnné age hodnota 20. [19]

3.3.3.5 *Extension functions*

Kotlin podobně jako C# má schopnost rozšířit třídu o novou funkcionalitu bez nutnosti využití dědičnosti nebo návrhového vzoru *Dekorátor*. Je toho docíleno pomocí speciálních deklarací nazývaných „extension functions“. [17]

Extension funkce je funkce, která může být zavolána jako člen třídy, ale je definována mimo tuto třídu.

```
fun String.lastChar(): Char {
    return get(length - 1)
}
```

Na příkladu výše lze vidět, že stačí napsat jméno třídy, kterou chceme rozšířit s tečkovou notací a zbytek je stejný jako u normální funkce. Je jedno, jestli je třída String napsána v Javě, Kotlinu nebo jiném JVM jazyku, není ani potřeba mít její zdrojový kód (pokud se jedná např. o knihovnu). Dokonce je možné bez problému přistupovat k metodám a proměnným třídy, kterou rozšiřujeme (pokud tedy nejsou private nebo protected, to by narušilo zapouzdření). K zavolání extension funkce je potřeba pouze vložit import s rozšiřující metodou. [6]

3.3.3.6 ***Object declarations***

„Object declarations“ neboli deklarace objektů je funkcionalita Kotlinu, která umožňuje deklarovat třídu, která bude mít jedinou instanci. Stačí použít klíčové slovo object místo class. Stejně jako třída objekt může obsahovat deklarace proměnných a metod. Pouze konstruktory nejsou povolené, inicializaci objektu je možné provést v init bloku (viz ukázka níže). Objekt je vytvořen při prvním použití s ohledem na bezpečnost při přístupu více vláken. Object declarations se dají použít například pro implementaci návrhového vzoru *singleton*. [20]

```
object Singleton {
    init {
        println("init complete")
    }

    val allEmployees = arrayListOf<User>()
}
```

3.3.3.7 *Coroutines*

V Kotlinu 1.1 byly představeny tzv. „Coroutines“ jakožto nový způsob psaní asynchronního kódu. Coroutines zjednodušují asynchronní programování, logika programu může být vyjádřena sekvenčně a o zbytek se postará knihovna. Jedná se zatím o experimentální funkcionalitu, což znamená, že se jejich používání může v budoucnu trochu změnit. Coroutines se dají chápat jako odlehčená vlákna (*thready*). Při vývoji dnešních aplikací se takřka nedá vyhnout spouštění úloh na více vláknech. Je možné se s tím setkat například při vykreslování animace načítání a zároveň stahování dat ze sítě. Stejně jako vlákna, coroutines mohou běžet paralelně, čekat na sebe navzájem a komunikovat. Na rozdíl od vláken jsou rychlejší, protože nevyžadují změnu kontextu (operace, kdy operační systém přepíná řízení mezi procesy). Jejich funkcionalita se velmi podobá async funkcím, které byly představeny Microsoftem v C# ve verzi 5.0. Coroutine je možné spustit pomocí funkcí launch a async. *[14][15]*

Funkce Launch

Povinným vstupním parametrem je funkce, jejíž obsah bude vykonán asynchronně. V nepovinných parametrech lze nastavit kontext a možnosti startu.

```
private fun main() {
    println("Start")

    // Odstartuje coroutine
    launch {
        delay(1000)
        println("Hello")
    }

    println("Stop")
}
```

V kódu výše je spuštěna coroutine pomocí funkce launch, která počká 1 sekundu a vypíše text „Hello“. Je zde použita funkce delay(), která pozastaví samotnou coroutine (bez blokování vlákna). Po čekání se coroutine vrátí na volné vlákno. Funkce main tedy vypíše nejdříve „Start“, poté „Stop“ a nakonec „Hello“. [16]

Funkce Async

Na rozdíl od funkce launch, která nevrací žádnou návratovou hodnotu, funkce async vrací instanci rozhraní Deferred, které má metodu await(), ta vrací výsledek dané coroutine. Je nutné zmínit, že metoda await() je označena klíčovým slovem suspend a lze ji zavolat pouze z coroutine.

```
private fun main() {
    println("Start")

    launch {
        val sum = async {
            5+5
        }

        println("Vysledek v async: ${sum.await()}")
    }
    println("Stop")
}
```

V kódu výše je ve funkci main nadefinována proměnná sum jejíž hodnota bude spočítána asynchronně pomocí coroutine. Funkce main vypíše „Start“, poté „Stop“ a nakonec „Vysledek v async: 10“. [16]

3.3.4 Kotlin a jeho použití

3.3.4.1 *Kotlin na straně serveru*

Programování na straně serveru je široký koncept. Nejčastěji v sobě zahrnuje následující typ aplikací:

- Webové aplikace, které vrací HTML stránky do webového prohlížeče
- Backend mobilních aplikací, které vystavují JSON API pomocí HTTP protokolu
- Mikroslužby, které komunikují s ostatními mikroslužbami pomocí RPC protokolu [5]

Vývojáři produkují tyto typy aplikací v Javě po mnoho let a za tu dobu vzniklo mnoho frameworků a technologií, které usnadňují práci při jejich vývoji. Např. Spring Boot, vert.x nebo JSF. Programovací jazyk Kotlin umožňuje používat naprostou většinu knihoven a frameworků napsaných v Javě (díky interoperabilitě s Javou viz. kap. 3.3.2.5). Je také možné napsat novou komponentu aplikace v Kotlinu, zatímco zbytek aplikace zůstane v Javě. Kotlin má i své vlastní frameworky, které usnadňují práci při vývoji server-side aplikací např. http://ktor.io/. [6]

3.3.4.2 *Kotlin na Androidu*

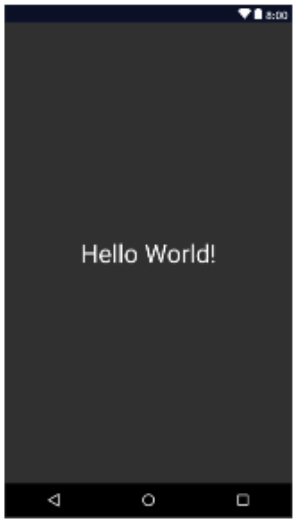
Tým vývojářů mobilního operačního systému Android oznámil během konference Google I/O 2017, že se Kotlin stává oficiálním jazykem pro vývoj aplikací pro Android. To znamená, že zatímco bylo stále možné vyvíjet aplikace pro Android v Javě, od tohoto okamžiku byl Kotlin plně podporován a Google zajistí, aby nové funkce Androidu, SDK a Android Studia (oficiální IDE po Android) byly v budoucnu dostupné pro tento jazyk. [10]

Vlastnosti Kotlinu kombinované se speciálním pluginem Kotlin Android Extensions pro kompilátor Android Frameworku měly za cíl proměnit vývoj na této platformě na mnohem pohodlnější a produktivnější úroveň. Běžné operace tedy vyžadují mnohem méně napsaného kódu nebo dokonce žádný kód. [6]

3.3.5 **Kotlin Android Extensions plugin**

Kotlin Android Extensions je plugin pro Kotlin. Tento plugin na pozadí generuje kód, který umožňuje zpřístupnit grafické komponenty definované v layoutu bez nutnosti volání metody findViewById(). Referenci na komponentu lze nyní jednoduše získat z kódu pomocí hodnoty jejího atributu id.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/welcomeMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Hello World!"
        android:textSize="36sp" />

</FrameLayout>
```



Obrázek 4: Definice layoutu v souboru activity_main.xml a jeho náhled, zdroj: autor

Aby bylo možné změnit text textového pole s id „welcomeMessage“ definovaného v xml na obr. 4 je potřeba napsat jeden z následujících řádků kódu:

Bez Android Extensions pluginu:

```
(findViewById<TextView>(R.id.welcomeMessage)).text = "Hello Kotlin!"
```

S Android Extensions pluginem:

```
welcomeMessage.text = "Hello Kotlin!"
```

Pro zpřístupnění textového pole v kódu s Android Extensions je ještě potřeba přidat speciální import z balíčku kotlinx.android.synthetic.main.*, to ale udělá Android Studio automaticky. Plugin tedy na pozadí volá metodu findViewById(), nedochází zde ovšem k volání této metody při každém přístupu k textovému poli. Při prvním volání dojde k uložení textového pole do cache a při dalším volání se vezme odtud. [12]

3.3.5.1 *Generátor implementace rozhrání Parcelable*

Od Kotlinu verze 1.1.4 byl do pluginu přidán generátor implementace pro rozhrání Parcelable. Třídy implementující toto rozhrání využívají serializační mechanismus Parcel optimalizovaný pro Android. Stačí přidat ke třídě anotaci @Parcelize a implementace rozhrání Parcelable bude vygenerována automaticky. [13]

4 Vlastní práce

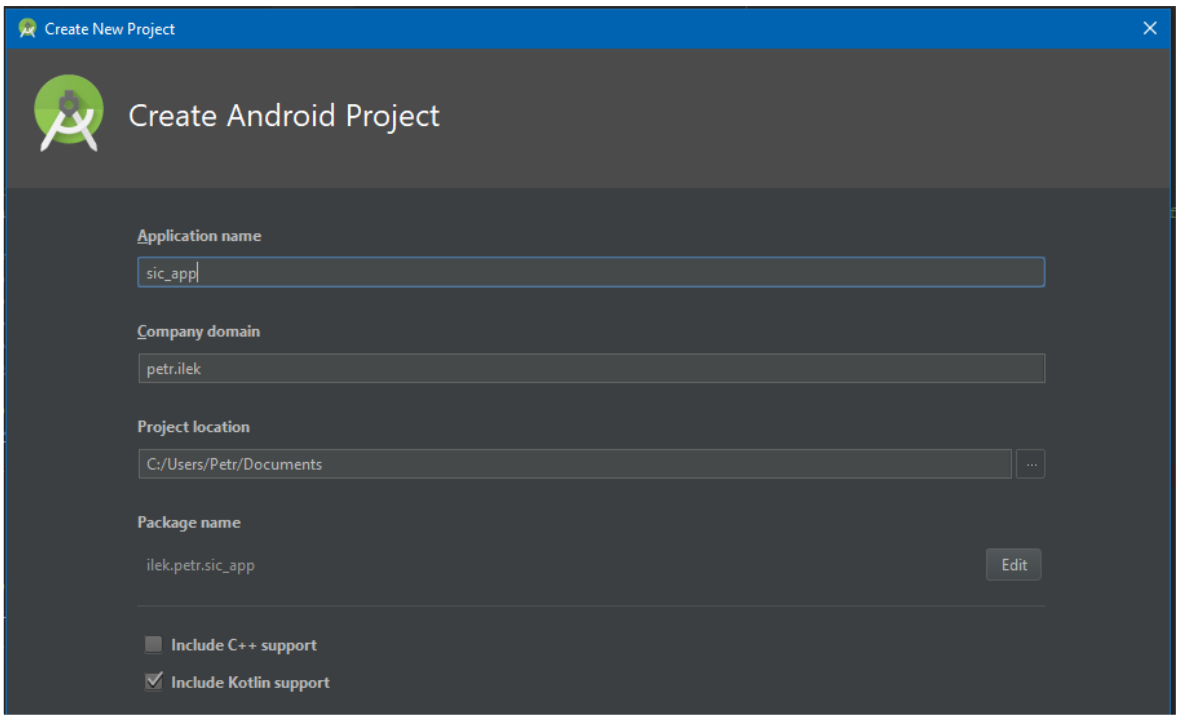
Praktickou částí této práce je implementace mobilní aplikace v jazyce Kotlin se shrnutím zkušeností z vývoje a porovnáním základních rozdílů oproti implementaci mobilní aplikace v Javě. Nejprve představím ukázkovou aplikaci. Poté popíši nejzajímavější části implementace a následně porovnám tyto části s implementací v Javě 8.

4.1 Ukázková aplikace

Ukázková mobilní aplikace slouží pro práci s univerzitním knihovním systémem Aleph. Aplikace je určena pro studenty univerzity a umožňuje spravovat své výpůjčky a rezervace, vyhledávat knihy v katalogu a přidávat si je do oblíbených. Aplikace běží na operačním systému Android.

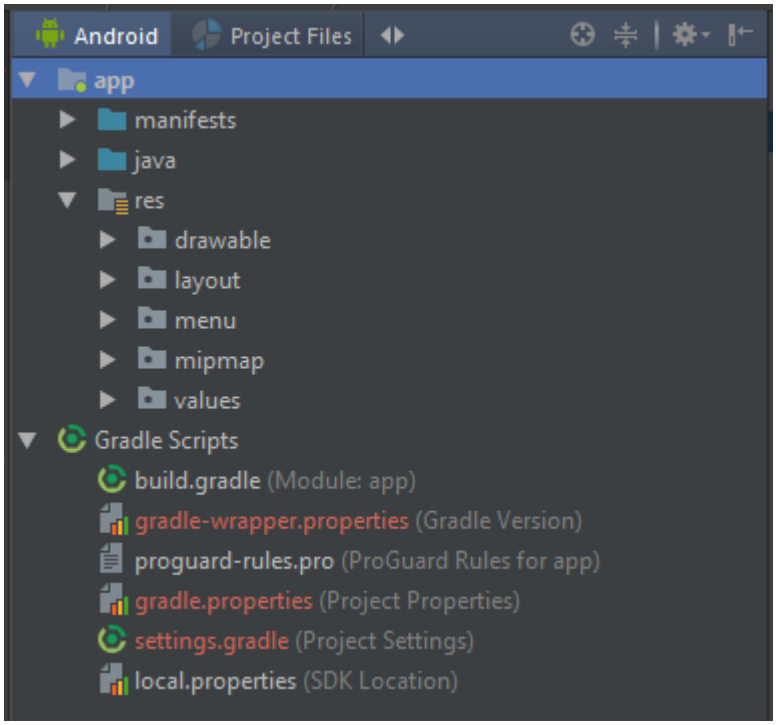
4.1.1 Založení projektu

Aplikace byla vyvíjena pomocí programovacího jazyku Kotlin 1.1.61 v oficiálním vývojovém nástroji pro Android – Android Studiu ve verzi 3.0, která již plně podporuje Kotlin, stačí pouze zaškrtnout volbu „Include Kotlin support“ při vytváření projektu.



Obrázek 5: Tvorba Kotlin projektu, zdroj: autor

4.1.2 Struktura projektu



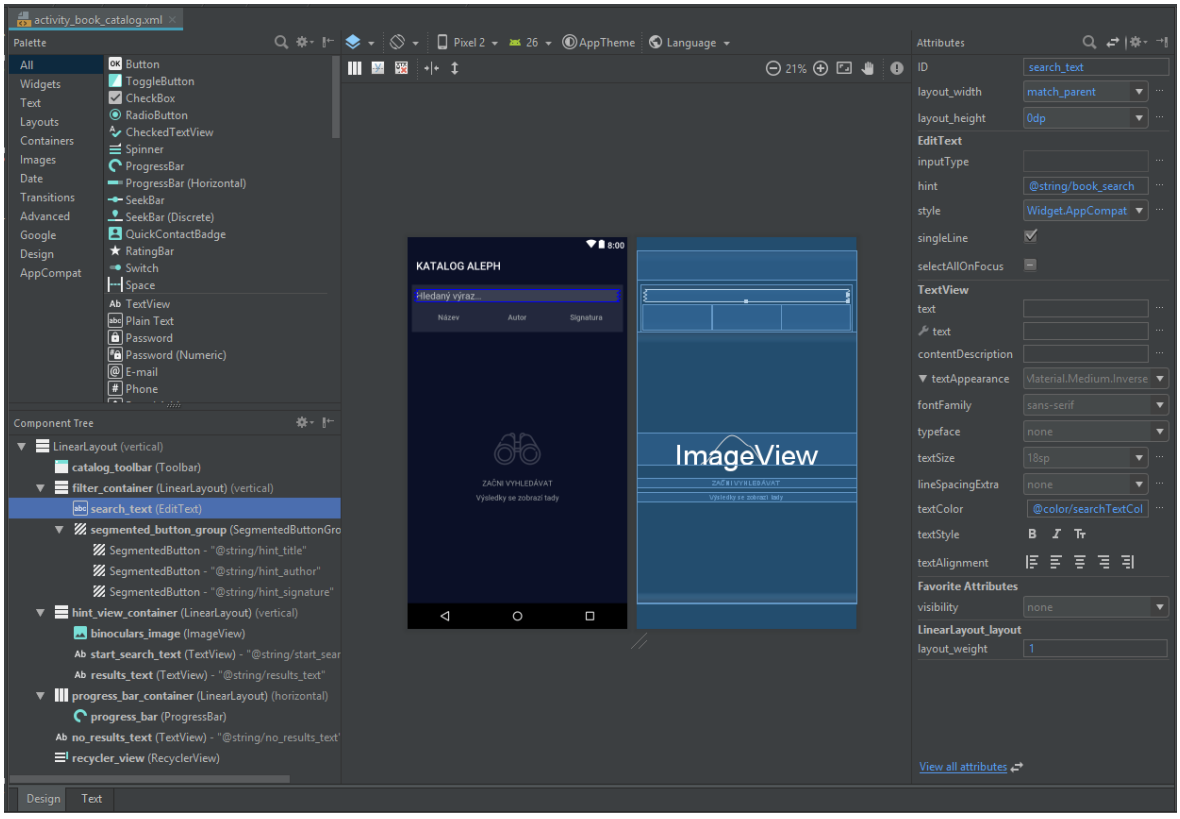
Obrázek 6: Struktura projektu, zdroj: autor

Projekt v Android Studiu se skládá z položek uvedených na obrázku výše. Nejdůležitější části projektu jsou popsány níže.

- **soubor AndroidManifest.xml** – popisuje základní informace o aplikaci jako je název, komponenty, a oprávnění
- **složka java –** obsahuje balíky se zdrojovými kódy v programovacím jazyku java (koncovka java) nebo kotlin (koncovka kt)
- **složka res –** obsahuje veškeré zdroje aplikace v následujících podsložkách:
  - **drawable –** veškerá grafika, která bude zobrazena obrázky, pozadí atd.
  - **layout –** struktura GUI popsaná v xml souborech
  - **menu** – struktury menu s možnostmi pro danou obrazovku definováno v xml souborech
  - **mipmap –** obsahuje ikony aplikace
  - **values –** obsahuje xml soubory s texty, barvami a styly aplikace
- **sekce Gradle Scripts –** se skládá z nastavení potřebného k buildu aplikace (nejčastěji se používá pro správu knihoven)

4.1.3 Tvorba grafického rozhraní

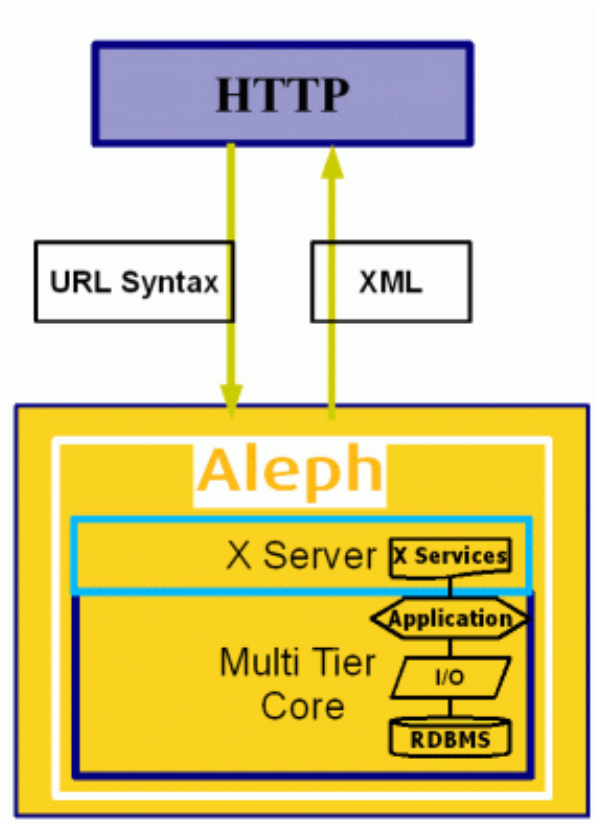
Každá obrazovka je nadefinována v XML souboru, do kterého se vkládají elementy s grafickými komponentami. Kořenový element určuje typ rozvržení (layout) obrazovky. Android Studio nabízí WYSIWYG editor pro tvorbu grafického rozhraní viz obrázek níže. Tento editor nabízí paletu s komponentami, které lze přesouvat do rozvržení obrazovky metodou drag and drop. Dále editor umožňuje nastavovat vlastnosti na vybrané komponentě, jako je např. její id, text, velikost písma, pozice atd. Mezi WYSIWYG editorem a obsahem XML se dá přepínat pomocí přepínání tlačítek „design“ a „text“ vlevo dole. Jelikož jsem použil Kotlin Android Extensions plugin (viz kapitola 3.3.5) ke zpřístupnění grafické komponenty v kódu stačí napsat její id, a poté s ní mohu pracovat jako se standartním objektem (nastavovat jí viditelnost, vlastnosti, listenery atd.).



Obrázek 7: Tvorba *GUI* katalogu

4.1.4 Komunikace s knihovním systémem

Aplikace komunikuje s knihovním systémem Aleph pomocí Aleph X-Serveru a jeho komponenty X-Services. Tato komponenta poskytuje data přes protokol http. Na vstupu přijímá URL syntax a na výstupu vrací XML s požadovanými daty, viz dokumentace na adrese: https://developers.exlibrisgroup.com/aleph/apis/Aleph-X-Services.



Obrázek 8: Komunikace s knihovním systémem Aleph, zdroj: [9]

Pro komunikaci s Aleph X-Services je potřeba mít schválenou IP adresu. Proto jsem naprogramoval pomocí technologie php pomocnou webovou aplikaci, která přijímá dotazy z mobilní aplikace a přesměruje je do knihovního systému. Tato aplikace běží na serveru s povolenou IP adresou a vyžaduje univerzitní přihlašovací údaje, které následně ověří v LDAPu univerzity. Teprve po úspěšné autentizaci uživatele přesměruje požadovaný dotaz z mobilní aplikace do knihovního systému. Aplikace komunikuje pomocí metody GET přes http protokol. Pro přesměrování jsou potřeba následující parametry:

- apiPath – požadavek, který bude přesměrován do X-Serveru
- username – univerzitní login
- password – heslo k univerzitnímu účtu

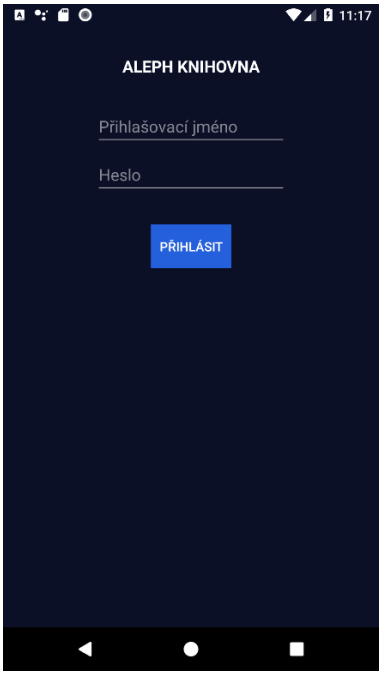
Veškerá komunikace s X-Serverem je zprostředkována pomocí rozhrání AlephApi viz níže.

```
interface AlephApi {

  suspend fun find(query: String, request: String,
                   credentials: UserCredentials): FindResult

  suspend fun present(offset: Int, numberSet: String,
                      credentials: UserCredentials): List<CatalogBook>

  suspend fun getItemData(docNumber: String,
                          credentials: UserCredentials): List<Item>

  suspend fun loginUser(username: String, password: String): User

  suspend fun holdRequest(docNumber : String, itemBarcode : String,
                          credentials: UserCredentials): Boolean

  suspend fun cancelHoldRequest(docNumber : String,
                                itemSequence : String,
                                sequence : String,
                                credentials: UserCredentials): Boolean

  suspend fun renewLoan(loan : LoanBook, credentials: UserCredentials): Boolean

}
```

Protože v Androidu nelze provádět síťové operace na hlavním vlákně jsou metody označeny klíčovým slovem suspend, mohou být tedy volané pouze z coroutine (viz kap. 3.3.3.7). Všechny výše uvedené metody vytvářejí dotaz do Aleph X-Service pomocí třídy URL a její metody readText, která vrátí odpověď jako řetězec. Obsahem tohoto řetězce je XML dokument, který je rozparsován pomocí XPathu.

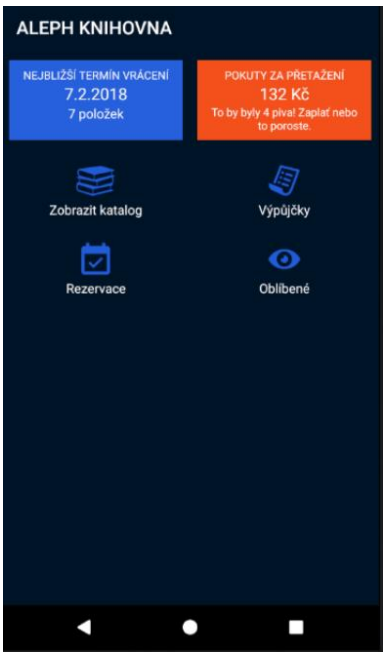
4.1.5 Obrazovky aplikace

Do aplikace se student přihlásí univerzitním loginem pomocí přihlašovací obrazovky na obr. 9.



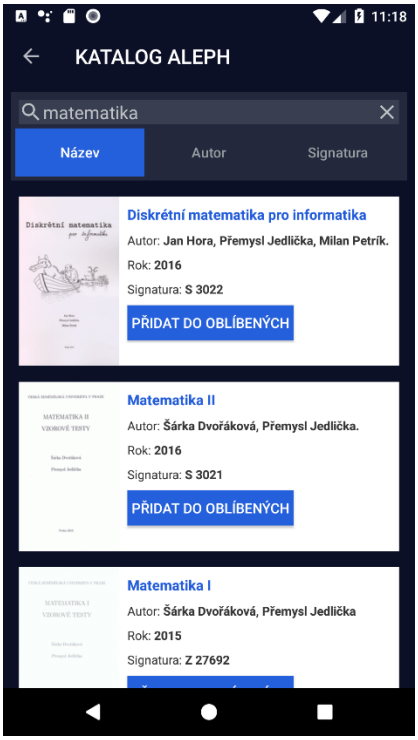
Obrázek 9: Přihlašovací obrazovka

Po přihlášení dojde ke zobrazení hlavního menu na obr. 10. Pokud má student nějaké výpůjčky dojde ke zobrazení notifikace s nejbližším datem vrácení. Pokud má student nezaplacené pokuty dojde ke zobrazení notifikace s celkovou částkou k zaplacení.



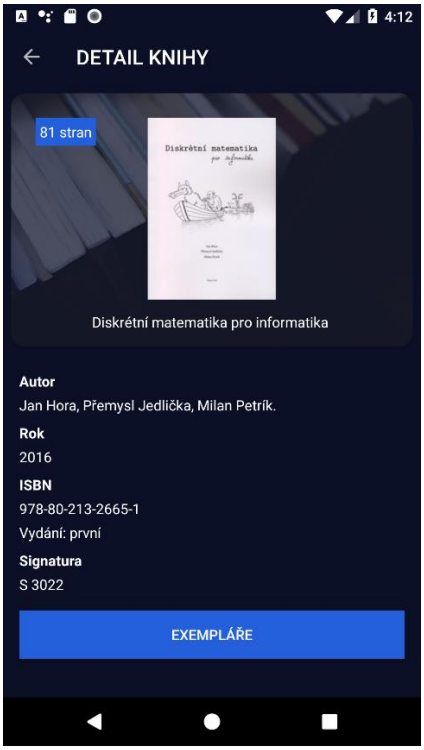
Obrázek 10: Hlavní menu

Po stisknutí tlačítka „Zobrazit katalog“ se zobrazí obrazovka s vyhledáváním na obr. 11. Student může vyhledávat knížky v univerzitním katalogu podle slov z názvu, autora nebo signatury. Každá nalezená kniha má tlačítko „přidat do oblíbených“.



Obrázek 11: Katalog knih

Při kliknutí na nalezenou knihu dojde ke zobrazení detailu knihy na obr. 12.

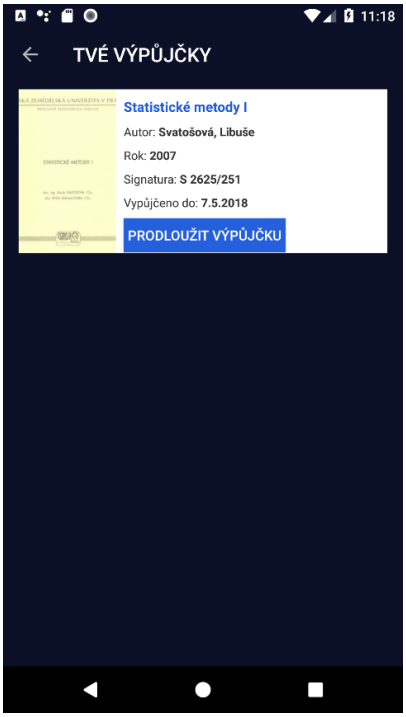


Obrázek 12: Detail knihy

Po stisknutí tlačítka „Exempláře“ dojde ke zobrazení jednotlivých exemplářů v knihovně, viz obr. 13.

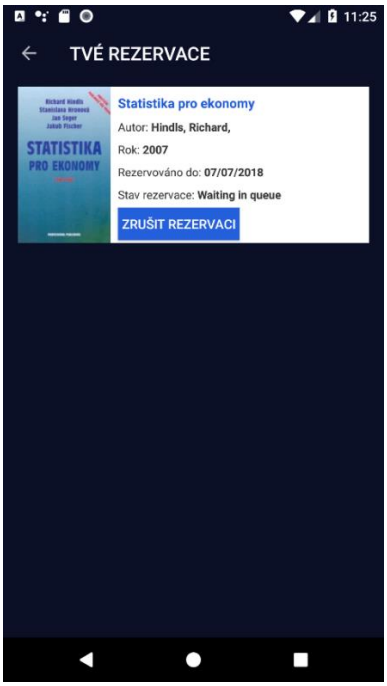
Obrázek 13: Exempláře

Po stisknutí tlačítka „Výpůjčky“ v hlavním menu dojde ke zobrazení výpůjček (obr. 14). Pomocí tlačítka „Prodloužit výpůjčku“ je možné výpůjčku prodloužit.



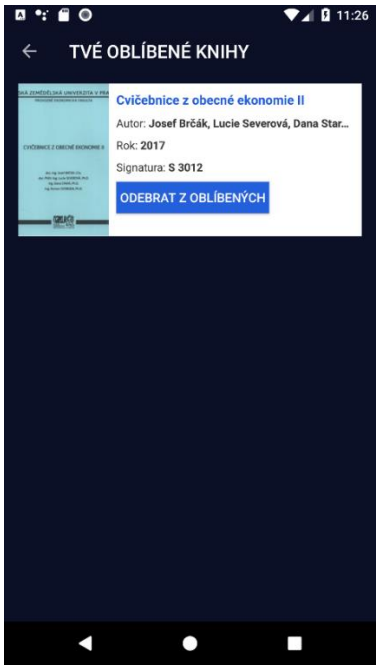
Obrázek 14: Výpůjčky

Po stisknutí tlačítka „Rezervace“ v hlavním menu se zobrazí rezervované knihy (obr. 15). Studentovi je umožněno rezervaci knihy zrušit pomocí tlačítka „Zrušit rezervaci“.



Obrázek 15: Rezervace

Po stisknutí tlačítka „Oblíbené“ v hlavním menu dojde ke zobrazení seznamu s oblíbenými knihami na obr. 16. Student si zde může oblíbenou knihu odebrat pomocí tlačítka „Odebrat z oblíbených“.



Obrázek 16: Oblíbené knihy

4.2 Zkušenosti z vývoje a porovnání s Javou

Tato kapitola popisuje, jak lze pomocí Kotlinu řešit nejčastější problémy při vývoji mobilní aplikace pro Android. Následně bude řešení těchto problémů porovnáno s řešením v programovacím jazyku Java 8.

4.2.1 Tvorba modelu

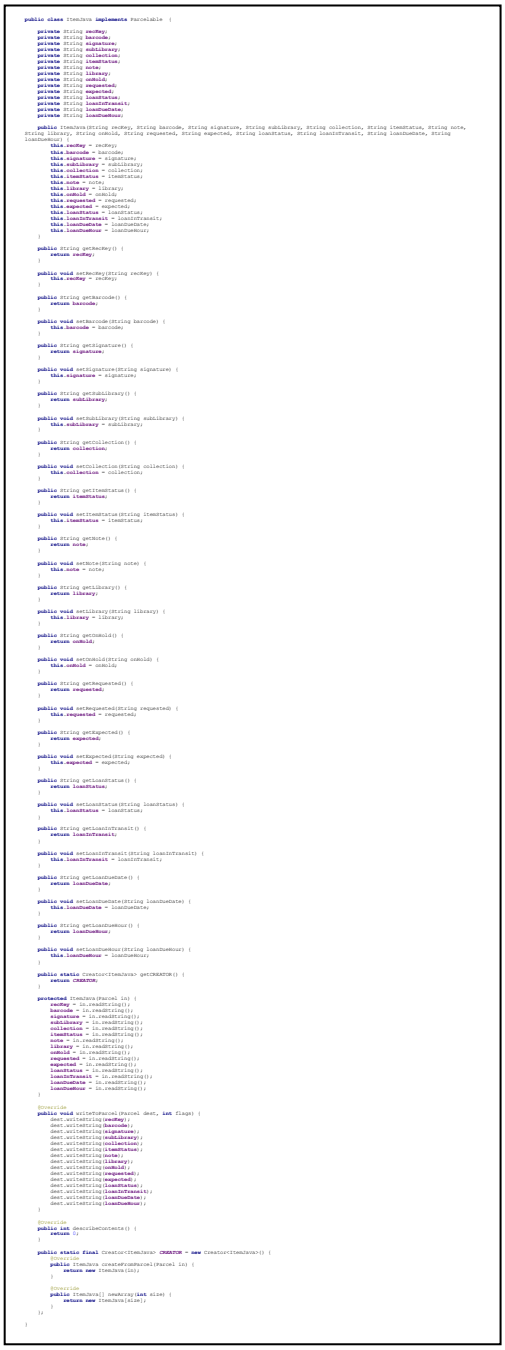
Při tvorbě modelu jsem využil data classes (viz kap. 3.3.3.3). V kódu níže je demonstrována implementace třídy Item, která představuje jeden exemplář knihy z katalogu.

```
@Parcelize
data class Item(val recKey: String,
                val barcode: String,
                val signature: String,
                val studySignature: String,
                val subLibrary: String,
                val collection: String,
                val itemStatus: String,
                val note: String,
                val library: String,
                val onHold: String,
                val requested: String,
                val expected: String,
                val loanStatus: String,
                val loanInTransit: String,
                val loanDueDate: String,
                val loanDueHour: String
) : Parcelable
```

Protože v aplikaci dochází k předávání instancí této třídy z obrazovky s katalogem knih na obrazovku detailu knihy, musí tato třída podporovat serializaci. Jelikož jsem použil Kotlin Android Extension Plugin, stačilo ke třídě přidat anotaci Parcelize (viz kap 3.3.5.1).

K dosažení podobné funkcionality v Javě by bylo potřeba napsat (nebo vygenerovat v IDE) kód o cca. 220ti řádcích. V Javě je totiž nutné napsat konstruktor, *gettery*, *settery* a implementovat rozhrání Parcelable. A pokud by bylo potřeba přetížit metody toString(), equals() a hashCode() byl by zdrojový kód zvětšen o dalších cca. 70 řádků. Všechny výše uvedené metody lze sice vygenerovat v IDE, ale pokud bych například chtěl později přidat do třídy Item novou proměnnou, znamenalo by to znovu vygenerovat všechny metody. V kotlinu by stačilo pouze danou vlastnost přidat do definice datové třídy. Pro představu hustoty Java kódu je dále uvedena ukázka implementace v Kotlinu (vlevo) s písmem o velikosti 9 a v Javě (vpravo) s velikostí písma 2.

```kotlin
@Parcelize
data class Item(
  val recKey: String,
  val barcode: String,
  val signature: String,
  val studySignature: String,
  val subLibrary: String,
  val collection: String,
  val itemStatus: String,
  val note: String,
  val library: String,
  val onHold: String,
  val requested: String,
  val expected: String,
  val loanStatus: String,
  val loanInTransit: String,
  val loanDueDate: String,
  val loanDueHour: String
) : Parcelable
```



4.2.2 Tvorba singletonu

Při implementaci logiky pro komunikaci s Aleph X-Serverem jsem použil návrhový vzor *singleton*. Kotlin pro tento problém nabízí jednoduché řešení pomocí tzv. „Object declarations“ (viz kapitola 3.3.3.6). Stačilo místo klasické třídy použít třídu s jedinou instancí pomocí klíčového slova object.

```
object AlephAPI
```

Ukázka volání metody loginUser() v Kotlinu:

```
AlephAPI.loginUser(user.username, user.password)
```

Pro vytvoření *singletonu* v Javě je potřeba napsat kód níže:

```
public final class AlephApiJava {

    public static AlephApiJava INSTANCE;

    private AlephApiJava() {
        INSTANCE = (AlephApiJava)this;
    }

    static {
        new AlephApiJava();
    }
}
```

Ukázka volání metody loginUser() v Javě:

```
AlephApiJava.INSTANCE.loginUser(user.getUsername(), user.getPassword())
```

V Javě je potřeba napsat o zhruba 10 řádek více při deklaraci *singletonu*. Pro zavolání metody loginUser je navíc nutno přistupovat na konkrétní statickou proměnnou s instancí třídy AlephApiJava.

4.2.3 Implementace rozhrání onClickListener

Pro vykonání vlastní logiky po stisknutí komponenty v GUI, je potřeba dané komponentě zaregistrovat *callback*. To se v Android SDK provádí pomocí rozhrání OnClickListener a implementace jeho metody onClick(View v). Vstupním parametrem této metody je instace třídy View, což je komponenta, na které bylo provedeno kliknutí (třída View je předek pro všechny GUI komponenty v Android SDK). Protože toto rozhrání má pouze jednu metodu, je možné použít v Kotlinu tzv. „*SAM conversions*“ z Javy 8 a nahradit toto rozhrání pouze implementací metody onClick() v lambda výrazu.

Ukázka v Kotlinu:

```
login_btn.setOnClickListener {
    //kod s logikou tlacitka
}
```

Ukázka v Javě 8:

```
login_btn.setOnClickListener(v -> {
    //kod s logikou tlacitka
});
```

V Kotlinu oproti Javě není potřeba v lambda výrazu definovat vstupní parametr, protože je to jediný parametr, který je možné zpřístupnit pomocí klíčového slova it.

4.2.4 **Asynchronní programování**

V Androidu není možné provádět některé typy operací na hlavním vlákně. Jedním z typů těchto operací jsou síťové operace. Pro přihlášení uživatele jsem potřeboval po vyplnění textových polí a stisknutí tlačítka „přihlásit“ provolat Aleph X-Server pro ověření uživatele a získání jeho dat. Toto provolání tedy nesmělo být vykonáváno na hlavním vlákně.

V Kotlinu šlo provolání jednoduše vyřešit díky *coroutines* – pomocí funkcí launch a async (viz kap. 3.3.3.7). Nadefinoval jsem konstantu user, do které pomocí funkce async asynchronně uložím výsledek metody loginUser(). Při jejím úspěšném vykonání uložím data uživatele a zobrazím obrazovku s menu. Volání metody await() pro zpřístupnění hodnoty proměnné user, musí být obaleno další coroutine pomocí funkce launch, protože metoda await je tzv. suspend funkce. Funkci launch volám s parametrem UI, protože zde přistupuji ke komponentám na GUI. Viz ukázka kódu dále.

```
login_btn.setOnClickListener {
    if (username.text.toString().isEmpty() ||
        password.text.toString().isEmpty()) {
        showAlertDialog(R.string.dialog_login, R.string.login_fill_all)
    } else {
        login_btn.visibility = GONE
        progress_bar.visibility = VISIBLE
        launch(UI) {
            try {
                val user = async {
                    AlephApiImpl.loginUser(username.text.toString(),
                                           password.text.toString())
                }
                UserData.saveUser(user.await(), this@LoginActivity)
                val intent = Intent(this@LoginActivity,
                                    AlephMenuActivity::class.java)
                ContextCompat.startActivity(this@LoginActivity, intent, null)
                finish()
            } catch (exception: Exception) {
                exception.printStackTrace()
                showAlertDialog(R.string.dialog_login, R.string.login_failed)
                progress_bar.visibility = GONE
                login_btn.visibility = VISIBLE
            }
        }
    }
}
```

V Javě se na Androidu pro asynchronní operace používá abstraktní třída AsyncTask s metodami doInBackground(), onPostExecute() a onCancelled(). Nadefinoval jsem třídu LoginUserTask, která dědí od třídy AsyncTask. Za účelem přistupování ke gui komponentám, bylo vytvořeno rozhrání LoginCallback s metodami OnLoginSuccess() a OnLoginFailed(). Tyto metody je možné implementovat pomocí anonymní třídy a přistupovat tedy k proměnným vnějším třídy – GUI komponentám. Instance tohoto rozhrání je předána v konstruktoru třídy LoginUserTask. Viz ukázka kódu dále.

```
login_btn.setOnClickListener(v -> {
    if (username.getText().toString().isEmpty() ||
        password.getText().toString().isEmpty()) {
        showAlertDialog(R.string.dialog_login, R.string.login_fill_all);
    } else {
        login_btn.setVisibility(GONE);
        progress_bar.setVisibility(VISIBLE);
        LoginUserTask loginUserTask = new LoginUserTask(
            new LoginUserTask.LoginCallback() {
            @Override
            public void onLoginSuccess(User user) {
                UserData.saveUser(user, LoginActivityJava.this);
                Intent intent = new Intent(LoginActivityJava.this,
                                           AlephMenuActivity.class);
                ContextCompat.startActivity(LoginActivityJava.this, intent,
                                            null);
            }

            @Override
            public void onLoginFailed() {
                showAlertDialog(R.string.dialog_login, R.string.login_failed);
                progress_bar.setVisibility(GONE);
                login_btn.setVisibility(VISIBLE);
            }
        });
        loginUserTask.execute(username.getText().toString(),
                              password.getText().toString());

    }
});
```

```
public class LoginUserTask extends AsyncTask<String, Void, User> {
    private LoginCallback afterLoginCallback;

    public LoginUserTask(LoginCallback afterLoginCallback) {
        this.afterLoginCallback = afterLoginCallback;
    }

    @Override
    protected User doInBackground(String... userCredentials) {
        try {
            return AlephApiJava.INSTANCE.loginUser(userCredentials[0],
                                                   userCredentials[1]);
        } catch (Exception e) {
            e.printStackTrace();
            cancel(true);
            return null;
        }
    }

    @Override
    protected void onPostExecute(User user) {
        afterLoginCallback.onLoginSuccess(user);
    }

    @Override
    protected void onCancelled() {
        afterLoginCallback.onLoginFailed();
    }

    public interface LoginCallback {
        void onLoginSuccess(User user);
        void onLoginFailed();
    }
}
```

V Kotlinu je kód zhruba dvakrát kratší než v Javě, protože nebylo potřeba vytvářet potomka třídy AsyncTask. Díky coroutines je logika běžící na pozadí napsána sekvenčně – nejsou zde tedy *callbacky* a kód je lépe čitelný. I když se jedná zatím o experimentální funkci Kotlinu, nenarazil jsem při použití coroutines na problémy.

5 Závěr

V první části práce byl představen Kotlin ve verzi 1.1 jako pragmatický, bezpečný a stručný programovací jazyk, který používá osvědčená řešení z jiných jazyků pro běžné úlohy v praxi. Byly popsány jeho prvky z funkcionálního programování, možnosti při asynchronním zpracování úloh a interoperabilita s programovacím jazykem Java.

Tyto znalosti byly aplikované při implementaci ukázkové aplikace v praktické části. Výsledkem je funkční mobilní aplikace pro operační systém Android, jejímž účelem je usnadnit studentům univerzity práci s katalogem knih v univerzitní knihovně. Na základě zkušeností z implementace ukázkové aplikace bylo popsáno, jak lze řešit nejčastější problémy při vývoji mobilních aplikací pro Android pomocí Kotlinu. Řešení těchto problémů bylo následně porovnáno s programovacím jazykem Java 8.

Z analýzy zdrojových kódů vyplývá, že použitím jazyka Kotlin lze dosáhnout stejné funkcionality pomocí menšího množství zdrojového kódu než v jazyce Java. To může vést k lepší čitelnosti a udržitelnosti kódu. Díky vzájemné interoperabilitě Kotlinu s Javou lze Kotlin používat v již odstartovaných projektech v Javě.

Programovací jazyk Kotlin se neustále vyvíjí a v budoucnu by měl podporovat kromě vývoje aplikací pro Java Virtual Machine, Android a Javascript také vývoj nativních aplikací pro macOS a iOS.

6 Seznam použitých zdrojů

[1] GOSLING, James a Henry MCGILTON. The Java Language Environment: Contents. *Oracle* [online]. 1996 [cit. 2017-09-10]. Dostupné z: http://www.oracle.com/technetwork/java/langenv-140151.html

[2] BEAL, Vangie, ed. Java. *Webopedia* [online]. [cit. 2017-09-10]. Dostupné z: https://www.webopedia.com/TERM/J/Java.html

[3] Definition of: Java Virtual Machine. *PcMag* [online]. [cit. 2017-10-9]. Dostupné z: https://www.pcmag.com/encyclopedia/term/45578/java-virtual-machine

[4] ROH, Štěpán. Java a JIT. In: *Štěpán Roh naživu leč ospalý* [online]. 2003 [cit. 2018-10-09]. Dostupné z: http://alivebutsleepy.srnet.cz/java-a-jit/

[5] FAQ. *Kotlin* [online]. 2017 [cit. 2017-10-09]. Dostupné z: https://kotlinlang.org/docs/reference/faq.html

[6] JEMEROV, Dmitry a Svetlana ISAKOVA. *Kotlin in Action, 1st Edition*, Manning Publications Co., 2017, 360 s. ISBN 9781617293290.

[7] ŠKARVADA, Libor. Co je to funkcionální programování. *Programujte.com* [online]. 2006 [cit. 2017-12-22]. Dostupné z: http://programujte.com/clanek/2006032503-co-je-to-funkcionalni-programovani/

[8] Null Safety. *Kotlin Programming Language: Reference* [online]. Czech Republic: Prague. JetBrains, 2017 [cit. 2018-02-05]. Dostupné z: https://kotlinlang.org/docs/reference/null-safety.html

[9] Ex Libris Developer Network. *Introduction to Aleph X-Services* [online]. Ex Libris [cit. 2018-02-05]. Dostupné z: https://developers.exlibrisgroup.com/aleph/apis/Aleph-X-Services/introduction-to-aleph-x-services

[10] LEIVA, Antonio. *Kotlin for Android Developers, 1st Edition*, CreateSpace Independent Publishing Platform, 2016, 200 s. ISBN 1530075610.

[11] *Android Developers* [online]. 2018 [cit. 2018-02-19]. Dostupné z: https://developer.android.com

[12] LEIVA, Antonio. *Kotlin Android Extensions: Say goodbye to findViewById* [online]. [cit. 2018-02-19]. Dostupné z: https://antonioleiva.com/kotlin-android-extensions/

[13] Kotlin Android Extensions. *Kotlin Programming Language: Reference* [online]. Czech Republic: JetBrains, 2017 [cit. 2018-02-05]. Dostupné z: https://kotlinlang.org/docs/tutorials/android-plugin.html

[14] SHEKHAR, Amit. What are Coroutines in Kotlin?. *Mindorks* [online]. 2017, 4. 10. 2017 [cit. 2018-02-19]. Dostupné z: https://blog.mindorks.com/what-are-coroutines-in-kotlin-bf4fecd476e9

[15] BUKROS, Adrian. Diving deep into Kotlin Coroutines. *Kotlin Development* [online]. 8. 10. 2017 [cit. 2018-02-19]. Dostupné z: https://www.kotlindevelopment.com/deep-dive-coroutines/

[16] Introduction to Kotlin Coroutines on the JVM. *Kotlin Programming Language: Reference* [online]. Czech Republic: Prague: JetBrains, 2017 [cit. 2018-02-19]. Dostupné z: https://kotlinlang.org/docs/tutorials/coroutines-basic-jvm.html

[17] Extensions. *Kotlin Programming Language: Reference* [online]. Czech Republic: Prague: JetBrains, 2017 [cit. 2018-02-19]. Dostupné z: https://kotlinlang.org/docs/reference/extensions.html

[18] Data Classes. *Kotlin Programming Language: Reference* [online]. Czech Republic: Prague: JetBrains, 2017 [cit. 2018-02-19]. Dostupné z: https://kotlinlang.org/docs/reference/data-classes.html

[19] SHEKHAR, Amit. Learn Kotlin- Destructuring Declarations. *Mindorks* [online]. 2017, 22. 5. 2017 [cit. 2018-02-19]. Dostupné z: https://mindorks.com/blog/learn-kotlin-destructuring-declarations

[20] BEYLS, Christophe. Kotlin singletons with argument. *Medium* [online]. 26 .8. 2017 [cit. 2018-02-20]. Dostupné z: https://medium.com/@BladeCoder/kotlin-singletons-with-argument-194ef06edd9e

[21] Kotlin Lambdas. *Programiz* [online]. [cit. 2018-03-03]. Dostupné z: https://www.programiz.com/kotlin-programming/lambdas

7 Přílohy

Příloha 1 - CD se zdrojovými kódy ukázkové aplikace