ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA TECHNOLOGIÍ A MĚŘENÍ

BAKALÁŘSKÁ PRÁCE

Tvorba aplikací na platformě .net pro OS Android

Jaroslav Malán **2013**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jaroslav MALÁN**
Osobní číslo:	**E10B0071P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Komerční elektrotechnika**
Název tématu:	**Tvorba aplikací na platformě .net pro OS Android**
Zadávající katedra:	**Katedra technologií a měření**

Zásady pro vypracování:

Prozkoumejte možnosti programování OS Android v jazyce C# nebo VisualBasic na plaform2 .NET Framework

1. Vytipujte projekty, které portují .NET Framework do OS Android
2. Vytvořte vzorové aplikace pomocí zvolených projektů.
3. Srovnejte způsob tvorby aplikací s nativním prostředím Androidu (Java) a Windows (klasický .NET Framework).
4. Funkčnost ověřte na emulátoru, v případě dostupnosti komerční verze také na reálném HW.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **20 - 30 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí bakalářské práce: **Ing. Petr Weissar, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání bakalářské práce: **15. října 2012**

Termín odevzdání bakalářské práce: **7. června 2013**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Ing. Vlastimil Skočil, CSc.
vedoucí katedry

V Plzni dne 15. října 2012

Abstrakt

Předkládaná bakalářská práce je zaměřena na vývoj aplikací pro mobilní zařízení se systémem Android. Aplikace pro operační systém Android byly předurčeny pro vývoj v programovacím jazyce Java, ale je také mnoho programátorů, kteří znají syntaxi jazyka C# a učení se základům jazyka Java by jim zabralo mnoho času. Projekt nazvaný Mono for Android byl představen právě z tohoto důvodu. Dalším jeho cílem je sjednocení vývoje softwaru pro všechny nejběžnější mobilní platformy. Problémem je, že každá mobilní platforma využívá pro tvorbu aplikací rozdílné programovací jazyky.

Tato práce se zabývá základními principy tvorby mobilních aplikací pro systém Android. Popisuje všechny nezbytné kroky pro započetí tvorby aplikací v nativním jazyce Java i v jazyce C#. Srovnává zdrojové kódy s totožnou funkcí v rozdílných programovacích jazycích a zkoumá výhody a nedostatky, které plynou z použití projektu Mono for Android pro vývoj aplikací pro systém Android. Pomocí jednoduchých testů výpočetního i grafického výkonu je porovnána výkonnost aplikací napsaných v jazyce C# s jazykem Java.

Projekt Mono for Android je kvalitně zpracovaný port jazyka C# pro operační systém Android. Po stránce výkonu se Mono for Android jeví jako více nebo alespoň stejně výkonná alternativa k programování v nativním jazyce Java. Jeho hlavní předností je možnost vývoje aplikace pro více mobilních platforem najednou. Avšak při neznalosti jazyka Java není Mono for Android vhodnou náhradou, protože je vždy nutné nastudovat tvorbu aplikací v nativní Javě, pro pochopení základních principů Androidu.

Klíčová slova

Mono for Android, Android, C#, Java, .NET prostředí, Xamarin, porovnání, vývoj mobilních aplikací, programování, objektově orientované programování

Abstract

The final thesis is focused on developing applications for Android mobile devices. Applications for operating system Android were supposed to be developed in programming language Java, but there are also many programmers which already know C# syntax and learning of basics Java language would take much time to them. A project called Mono for Android was introduced because of this fact. The next goal of the project is to unite software development for all the most common mobile platforms. The problem is that every mobile platform uses different programming languages for software development.

This thesis deals with the basic priciples of creating mobile applications for Android. It describes all necessary steps to begin creating applications in native Java and in C#. It compares source codes with identical functions written in different programming languages and examines the andvantages and disadvantages that arose from the use of the project Mono for Android to develop applications for Android. Simple tests of computing and graphics performance are used to compare the efficiency of the applications written in C# and Java.

The project Mono for Android is well-prepared port of C# language for operating system Android. The performance of Mono seems to be at least as efficient as the Java language programming is. Its main advantage is the possibility of developing applications for multiple mobile platforms at once. The Mono for Android is not a suitable substitute in case of lack Java knowledge because it is always necessary to study the development of native applications in Java, for an understanding of basic Android principles.

Key words

Mono for Android, Android, C#, Java, .NET Framework, Xamarin, comparation, mobile application development, programming, object oriented programming

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

.....

podpis

V Plzni dne 7.6.2013 Jaroslav Malán

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Petrovi Weissarovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce a také za zprostředkování příležitosti tvorby komerční aplikace pro systém Android u společnosti Veřejná informační služba IT spol. s.r.o. Dále bych rád v této práci poděkoval celé mé rodině, která mi byla v průběhu celého mého studia nepostradatelnou oporou.

Obsah

OBSAH **7**

SEZNAM SYMBOLŮ A ZKRATEK **8**

ÚVOD **9**

1 OPERAČNÍ SYSTÉM ANDROID A VÝVOJ APLIKACÍ **10**

1.1 ZÁKLADNÍ INFORMACE O VÝVOJI APLIKACÍ PRO ANDROID 11
1.2 POTŘEBNÉ NÁSTROJE PRO VÝVOJ - JAZYK JAVA 12
1.3 POTŘEBNÉ NÁSTROJE PRO VÝVOJ - JAZYK C# 13
1.4 SOFTWARE DEVELOPER KIT – JAVA I C# 13
1.5 ARCHITEKTURA APLIKACÍ PRO ANDROID 14
1.5.1 Activity *15*
1.5.2 View *19*
1.5.3 Jednoduchá aplikace využívající Activity a View *20*

2 MONO FOR ANDROID **25**

2.1 ARCHITEKTURA MONO FOR ANDROID 25
2.2 VÝKON MONO FOR ANDROID 27
2.2.1 Aplikace testující výpočetní výkon jazyků C# a Java *27*
2.2.2 Aplikace testující grafický výkon jazyků C# a Java *29*
2.3 OMEZENÍ MONO FOR ANDROID 31
2.4 VYDÁNÍ NOVÉ VERZE XAMARIN 2.0 31

3 VZOROVÁ APLIKACE PRO ZÁPIS DO SOUBORU NA SD KARTU **32**

ZÁVĚR **36**

SEZNAM LITERATURY A INFORMAČNÍCH ZDROJŮ **37**

PŘÍLOHY **1**

Seznam symbolů a zkratek

ACW ..... Android Callable Wrappers

ADB ..... Android Debug Bridge

ADT ..... Android Development Tools

API ..... Application programming interface

AVD ..... Android Virtual Devices

HTML ..... HyperText Markup Language

IDE ..... Integrated Development Environment

MCW ..... Managed Callable Wrappers

MS ..... Microsoft

SD ..... Secure Digital (paměťová karta)

SDK ..... Software Devepment Kit

SE ..... Sony Ericsson

USB ..... Universal Serial Bus

VIS-OMNO ..... Veřejná Informační Služba – Objednávací Místo Na Období

WYSIWYG ..... What You See Is What You Get

XML ..... Extensible Makup Language

Úvod

Předkládaná bakalářská práce se zabývá možností tvorby aplikací pro mobilní zařízení se systémem Android na platformě .NET v programovacím jazyce C#. Ačkoliv byl operační systém Android od svého počátku navržen pouze pro aplikace napsané v programovacím jazyce Java, společnost Xamarin nabídla svým projektem Mono for Android, možnost vývoje v jazyce C#. Tato skutečnost může mnohým programátorům, kteří nemají zkušenosti s nativní Javou, ušetřit mnoho práce, kterou by museli věnovat studiu jazyka Java.

Tuto práci jsem zpracoval na základě zkušeností s programováním komerční aplikace VIS-OMNO pro systém Android v nativním jazyce Java. Tato aplikace byla určena pro uskutečnění objednávek jídel ve školních jídelnách na terminálu s operačním systémem Android ve verzi 2.3 a byla určena pro společnost Veřejná informační služba IT spol. s.r.o.

V této práci se zabývám nejen popisem všech nezbytných kroků pro realizaci jednoduchých aplikací v jazyce C# pro operační systém Android, ale také zde hodnotím výhody a nevýhody, které celý projekt Mono for Android vývojářům poskytuje. Pro možnost porovnání tvorby aplikací v jazyce Java a C# je nutné znát základy tvorby aplikací pro operační systém Android. Tyto základy jsou popisovány v první části práce, kde je čtenář seznámen se základními stavebními prvky aplikace včetně životního cyklu aktivity, jehož znalost je nezbytná pro efektivní tvorbu aplikací.

Projekt Mono for Android za tento rok prošel mnoha změnami, které se týkají nejen zlepšování samotného softwaru Mono for Android, ale také musel nutně reflektovat všechny změny, kterými prošly originální Android Java knihovny v návaznosti na velmi časté vydávání nových verzí operačního systému Android. Proto musím upozornit, že popsané metody jsou platné pro rok 2013 a v případě problémů, je vhodné navštívit stránky *http://xamarin.com/*, kde se nachází aktuální rady a dokumentace celého projektu.

1 Operační systém Android a vývoj aplikací

V posledních letech zaznamenal trh mobilních zařízení nevídaný rozvoj a to především díky příchodu tzv. chytrých mobilních telefonů. V současné době dominují trhu s mobilními zařízeními operační systém Android od společnosti Google, operační systém iOS od společnosti Apple a také, v neposlední řadě, operační systém Windows Phone v poslední verzi 8. V této práci jsem se zaměřil na operační systém Android, který se mi především díky své otevřenosti jeví jako nejpraktičtější operační systém pro vývoj aplikací s dotykovým ovládáním. Dalším důvodem proč se zaměřit na vývoj aplikací právě pro operační systém Android, je jeho rozšířenost. Dle posledního vyjádření Googlu bylo aktivováno do března 2013 přes 750 milionů zařízení a bylo staženo celkem přes 25 miliard aplikací pomocí služby Google Play [1]. Z těchto čísel je jasně vidět, že Android je skutečně živým operačním systémem a v blízké budoucnosti bude mít jedno z dominantních postavení na trhu.

Android je rozsáhlá open source platforma, díky čemuž je systém distribuován v mobilních zařízeních všech cenových kategorií a mnoha rozličných značek. Samotné jádro operačního systému vychází z jádra systému Linux. V současné době se o vývoj nových verzí stará konsorcium Open Handset Alliance, které si dává za cíl progresivní rozvoj mobilních technologií, které umožní nižší náklady na vývoj a distribuci mobilního softwaru. Celý tento systém je koncipován tak, aby respektoval omezené možnosti, které plynou ze samotných mobilních zařízení. Mezi tato omezení patří například omezení výdrží baterie, menší výkon a méně operační paměti, než je tomu u desktopových počítačů [2].

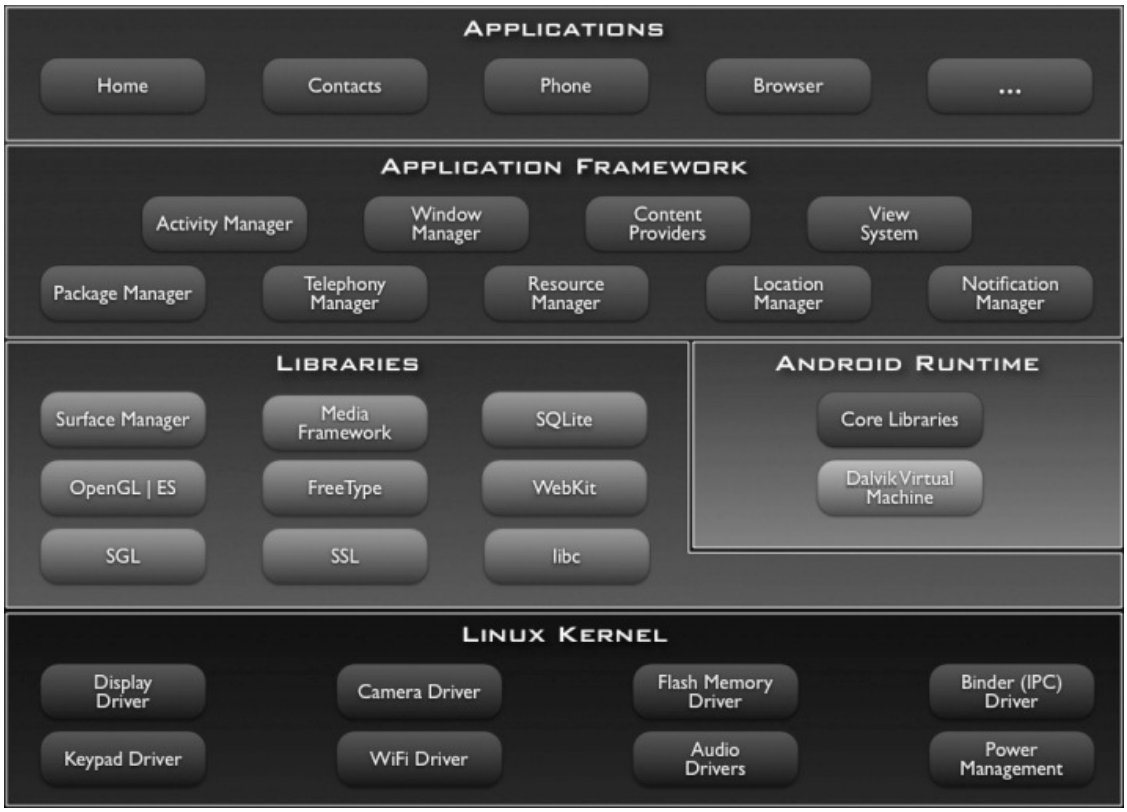
Samotný operační systém Android je pro svoji velkou oblíbenost stále ve vývoji. Za velmi krátkou dobu od vydání první verze systému Android v České republice, které se uskutečnilo v roce 2009, proběhlo velmi mnoho zásadních změn a rozdíly mezi jednotlivými verzemi, jak po stránce vývoje aplikací, tak i při běžném užívaní. Tyto všechny rozdíly mohou být propastné. I proto doporučuji všem, kteří mají zájem se blíže seznámit s vývojem aplikací pro operační systém Android, aby čerpali informace především z oficiálních stránek *http://developer.android.com*. Na těchto stránkách se nachází mnoho užitečných rad a informací včetně kompletní Java dokumentace, která je nepostradatelnou příručkou pro všechny Android vývojáře.

1.1 Základní informace o vývoji aplikací pro Android

Standardně jsou Android aplikace vyvíjeny v programovacím jazyce Java. Díky tomu jsou všechny potřebné nástroje pro zahájení programování zcela zdarma. Výhodou také je, že jazyku Java se v poslední době věnuje mnoho pozornosti a patří mezi nejrozšířenější programovací jazyky na světě. *„Java na rozdíl od řady programovacích jazyků nebyla ani dílem akademického výzkumu a vývoje, ani živelného nabalování se standardizací ex-post. Vznikla jako profesionální dílo poměrně přísně řízeného komerčního vývoje.“* [3]

Pokud chceme vyvíjet aplikace pro Android, tak je však nejdříve nutné se seznámit s architekturou samotného systému. Jak bylo již dříve zmíněno, platforma je postavena na linuxovém jádru, které je doplněno o řadu C/C++ knihoven. Mezi tyto knihovny se řadí například knihovny pro práci s médii, knihovny grafického enginu, SQLite knihovny a mnoho dalších.

Celý systém je nativně koncipován pro tvorbu aplikací v jazyce Java, proto by se dalo předpokládat, že na systému Android poběží Java Virtual Machine. To však není pravdou a samotný bytecode je vykonáván upravenou verzí Java Virtual Machine, která se nazývá Dalvik Virtual Machine. Díky vyvinutí a následnému použití Dalviku společností Google pro svůj operační systém Android podala společnost Oracle, která vlastní většinu licencí spojovaných s jazykem Java, žalobu na společnost Google. V březnu 2012 však soud rozhodl, že nedošlo k porušení patentu ze strany společnosti Google a společnost Oracle tedy nemá nárok na odškodnění [4].



Obr. 1: Architektura systému Android (převzato z [5])

1.2 Potřebné nástroje pro vývoj - jazyk Java

Pro započetí vývoje aplikací pro Android v jazyce Java je samozřejmě nejdříve nutné stáhnout tzv. Software Developer Kit, bez kterého by nebylo možné začít programovat. Zmíněné SDK je možné stáhnout z oficiálních stránek pro Android vývojáře na adrese *http://developer.android.com* . Na těchto stránkách se také nachází mnoho užitečných rad a návodů včetně kompletní dokumentace, bez které se při vývoji komplexnějších aplikací nedá dozajista obejít. Díky snaze Googlu o co největší rozšíření systému Androidu je SDK nabízeno pro všechny tři nejběžnější operační systémy, kterými jsou Windows, Mac i Linux.

Samozřejmostí je také nutnost stažení a instalace vhodného vývojového prostředí tzv. IDE. Původně bylo jedinou možností vývojové prostředí Eclipse, avšak s postupem času byla přidána i možnost vývoje Android aplikací v prostředí Netbeans. V této práci jsem zvolil vývojové prostředí Eclipse a to i proto, že Android plugin pro vývojové prostředí Elipse je přednostně vyvíjen. Všechny sepsané postupy týkající se programování aplikací v jazyce Java budou z tohoto předpokladu vycházet.

Posledním nutným předpokladem pro započetí programování Android aplikací v prostředí Eclipse je stažení výše zmíněného oficiálního ADT pluginu, který je k nalezení na oficiální stránce pro vývojáře, kde je ke stažení Android Software Developer Kit. Tento plugin nám mimo jiné umožní spuštění emulátoru, který je součástí SDK a tím nám umožní testování napsané aplikace v PC i bez nutnosti vlastnit skutečné Android zařízení.

1.3 Potřebné nástroje pro vývoj - jazyk C#

Mono for Android je jedinečné svojí snadnou instalací. Z hlavní stránky společnosti Xamarin je možné stáhnout jediný balíček, který by v ideálním případě měl po spuštění automaticky nainstalovat všechny potřebné součásti pro začátek programování. Součástí balíčku je i částečně upravené SDK, pomocí kterého je možno nainstalovat Mono emulátory, které se však nijak výrazně neliší od klasických emulátorů v běžném SDK a lze je vzájemně kombinovat. V průběhu instalace je také možné naistalovat zásuvný modul pro MS Visual Studio. Důležité však je zkontrolovat, zda není v době instalace spuštěné.

1.4 Software Developer Kit – Java i C#

Ačkoliv byl již Android Software Developer Kit stažen a nainstalován, je nutné do něj doinstalovat mnoho dalších volitelných součástí, které nejsou obsaženy v základní instalaci. Proto je nutné ze složky SDK spustit program *android*, kterým se spouští Android SDK a AVD Manager. Skrze tento manager je možné stahovat jednotlivé komponenty, jako jsou například nové verze API pro nové operační systémy Android, vzorové aplikace se zdrojovými kódy i více verzí různých emulátorů.

Mezi hlavní a nejdůležitější komponenty SDK patří samotné platformy Androidu. Tyto platformy odpovídají produkčním platformám, které jsou dostupné na skutečných zařízeních. V platformách SDK jsou obsaženy systémové knihovny, systémový obraz, skiny emulátoru, ukázky kódu a jiné zdroje specifické pro platformu.

Pokud je plánováno použít pro ladění napsaných aplikací skutečné zařízení Android namísto emulátorů, je nutné doinstalovat USB ovladač zařízení. Tento ovladač je nutné stáhnout prostřednictvím výše zmíněného *manageru*, kde se standardně nachází pod názvem „USB Driver“. Postup samotné instalace ovladače může činit problémy, proto zde pro maximální zjednodušení poskytuji krokovaný návod pro operační systém Windows 7.

Instalace USB ovladače Android zařízení (Win7):

1. připojíme Android zařízení k počítači pomocí USB portu
2. v Ovládacích panelech systému Windows otevřeme Správce zařízení
3. vyhledáme připojené zařízení a pravým tlačítkem myši vybereme Aktualizovat software ovladače
4. zvolíme možnost „Vyhledat ovladač v počítači“
5. zvolíme možnost „Vybrat ovladač ze seznamu“
6. zvolíme možnost „Z disku…“
7. nyní určíme cestu k ovladači se standardním umístěním ve složce „…\android-sdk\extras\google\usb_driver“
8. následně vybereme ze seznamu ovladačů variantu „Android Debug Bridge“ se standardním umístěním na prvním místě seznamu

Složka SDK s názvem Platoform-tools obsahuje vývojové nástroje, jakým je například velmi důležitý program *adb* neboli Android Debug Bridge. Tento program zprostředkovává přenos dat z PC do Android zařízení a to prostřednictvím USB rozhraní. Pozorný čtenář si mohl všimnout, že zmíněný název Android Debug Bridge byl již obsažen v instalaci USB ovladače zařízení, který je koncipován právě pro obsluhu tímto programem. Dalším programem v této složce je program *aapt*, které se stará o vytváření a modifikaci aplikačních balíků.

1.5 Architektura aplikací pro Android

Pokud jste se již setkali s instalačním souborem aplikace pro operační systém Android, tak jistě víte, že tyto soubory jsou zakončeny koncovkou *apk*. Ačkoliv to není na první pohled patrné, celý soubor s touto koncovkou je vlastně ZIP balík a je možné ho zcela běžně rozbalit a prozkoumat obsah.

Aplikace spuštěné na Android zařízení mají každá svou vlastní instanci virtual machine a vychází ze čtyř základních komponentů, kterými jsou: activity, service, broadcast receiver a content provider.

1.5.1 Activity

Jedná se o základní kámen všech aplikací. Aktivita vytváří základní visuální výstup, kdy každá jedna aktivita by měla odpovídat právě jedné obrazovce na zařízení. Ačkoliv je možné využít pouze jednu aktivitu a měnit její obsah, je tento postup nevhodný a neefektivní! Níže je zdrojový kód v jazyce Java, který je automaticky vygenerován jako hlavní vstupní aktivita při založení nového projektu ve vývojovém prostředí.

```
package com.example.helloword;

import android.os.Bundle;
import android.app.Activity;


public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Zdrojový kód začíná příkazem *package*, který popisuje umístění souboru této třídy v hierarchii tříd daného projektu. Využití již hotových tříd včetně jejich metod, či jejich dědění je umožněno jejich „přibalením“ ke třídě pomocí následujícího příkazu *import*. V tomto kódu je importována třída *android.os.Bundle*, která umožní předání objektu s různorodým obsahem metodě onCreate. Následně je importována třída *android.app.Activity*, která umožní zdědit vlastnosti třídy Activity nutné pro běh aplikace na Androidu. Následně je vytvořena samotná třída s veřejným obsahem a pomocí příkazu *extends* dědí vlastnosti a metody od dříve naimportované Activity. Příkaz *@Override* definuje, že chceme přepsat metodu onCreate zděděnou od Activity svojí vlastní metodou onCreate. Metoda onCreate je zásadní částí každé aktivity, protože její obsah slouží jako posloupnost příkazů, které se vykonají po spuštění aktivity. Je to tedy místo, odkud začínají všechny příkazy, které se mají v dané aktivitě vykonat. Příkazem *super.onCreate(savedInstanceState)* je spuštěna metoda onCreate(…) rodičovské třídy a po jejím vykonání se pokračuje následujícím kódem v naší metodě onCreate, kterým je v našem případě *setContentView(R.layout.activity_main)*. Poslední zmíněný příkaz je použit k přiřazení určitého visuálního rozvržení obrazovky. To znamená, že uvnitř projektu také existuje vytvořené rozvržení obrazovky s názvem *aktivity_main.xml*, které obsahuje mimo jiné prvek textu, který může mít nastavený text například na běžnou frázi „Hello word“. Z toho vidíme, že především vizuální prvky jako jsou

zobrazené texty, tlačítka či obrázky nemusí být nutně součástí zdrojového kódu, ale je možno je graficky zpracovat v samostatném souboru ve formátu xml. Samozřejmě lze všechny visuální prvky vytvořit programováním přímo ve třídě naší aktivity, ale z vlastní praxe mohu konstatovat, že tato varianta je přínosná pouze za předpokladu vytváření dynamicky se měnícího obsahu. Takovýmto obsahem může být například seznam s dopředu neznámým množstvím položek. V opačném případě je časově podstatně méně náročné a i více přehledné využití xml souborů s rozvržením visuálních prvků a jejich následné použití v kódu příkazem setContentView(…). Níže je zdrojový kód, který má totožnou funkci, avšak nyní v jazyce C#.

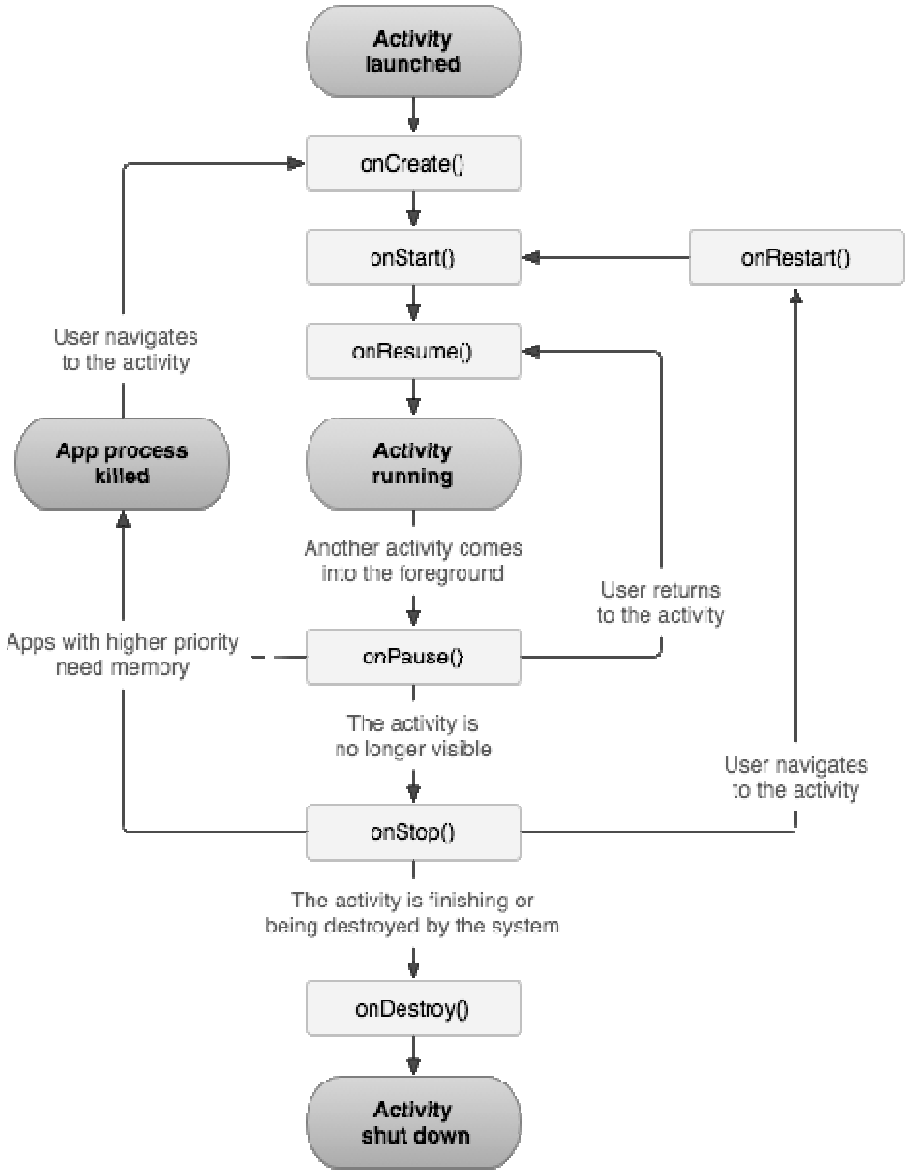
```
using Android.OS;
using Android.App;

namespace HelloWord
{
    [Activity(Label = "HelloWord", MainLauncher = true, Icon = "@drawable/icon")]
    public class Activity1 : Activity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView(Resource.Layout.Main);
        }
    }
}
```

Jak můžeme vidět, tak zásadní rozdíly jsou pouze v syntaxích jazyků Java a C#. Příkaz *using* nahrazuje *import*. Importované třídy se však mohou lišit velikostí prvních písmen a také hierarchie tříd může být rozdílná. Tyto rozdíly však nejsou nijak zásadní, a pokud znáte název třídy k importování, tak pomocí dokumentace Mono for Android není nalezení stejné třídy v C# nijak složité. Příkaz *namespace* nahrazuje Javovský *package*. Odkaz na rodiče je v jazyce C# *base* na rozdíl od *super* v jazyce Java. Zvláštností je zde automaticky vygenerovaný řádek v hranatých závorkách. Uvnitř tohoto příkazu se nastaví popisek aplikace, zda se má tato aktivita skutečně spouštět po startu aplikace a také je zde nastavena ikona, pod kterou je možné spustit tuto aplikaci.

Celý způsob spouštění aplikace včetně všech základních nastavení, jako jsou požadavky na minimální verzi zařízení, hlavní vstupní aktivitu při zapnutí aplikace, či povolení nutná k přístupu k prostředkům bezdrátové sítě, je obsažen v souboru Manifest, který se nachází vygenerován v kořenovém adresáři vytvořeného projektu pod názvem *AndroidManifest.xml*. K úpravám tohoto souboru je možno použít grafického editoru. Pouze u projektů vytvořených v Mono je nutné vygenerovat manifest otevřením nastavení projektu pravým tlačítkem myši.

1.5.1.1 Životní cyklus aktivity



Obr. 2: Životní cyklus aktivity (převzato z [6])

Aktivity pracují na principu komínového skládání se na sebe, přitom vždy ta nejvýše postavená aktivita je zobrazena a po jejím ukončení se zobrazí ta aktivita, která je položena hned pod ní. Na obrázku číslo 2 je možné vidět kompletní cyklus života aktivity.

Čtyři základní stavy života aktivity [6] :

1. Pokud je aktivita v popředí, zobrazená na obrazovce a je aktivní a běžící.
2. Pokud je aktivita částečně překryta jinou aktivitou, která však zaplní pouze část obrazovky. Potom je aktivita, která vyplňovala celou obrazovku pozastavena, avšak je stále živá a uchovává všechny proměnné a je připravena k opětovnému spuštění. Pouze při velmi zásadním nedostatku paměti může dojít k ukončení pozastavené

aktivity.

3. Pokud je aplikace kompletně zakryta jinou aktivitou, tak je zastavena. Aplikace v zastaveném stavu stále uchovává informace, ačkoliv není viditelná. Zastavené aktivity mohou být často ukončeny, pokud má systém nedostatek volné paměti.
4. Pokud je aplikace zastavena nebo i případně pozastavena, může dojít k ukončení aktivity ze strany systému Android, který takto činí v případě nedostatku paměti. Při následném spuštění stejné aktivity musí všechny příkazy proběhnout od začátku znova.

Na obrázku č. 2 je možno vidět, jak probíhá celý životní cyklus aktivity. Prvky v hranatých obdélnících popisují metody, které je možno implementovat do svého kódu. Pomocí těchto metod můžeme provádět příkazy, které jsou aktivované přechody aktivity mezi jednotlivými stavy. Oproti tomu v oválných prvcích jsou popsány všechny čtyři základní stavy, do kterých se může aktivita dostat.

V zásadě jsou tři běžné smyčky, které je vhodné hlídat v životním cyklu aktivity [6].

- **Celý životní cyklus** aktivity probíhá mezi metodami onCreate(Bundle) a je ukončen metodou onDestroy(). V metodě onCreate(…) se spustí většina příkazů, které inicializují nezbytné proměnné pro běh aktivity. V metodě onDestroy() se standardně ukončují například vlákna, která byla spuštěna metodou onCreate(…) .
- **Viditelný životní cyklus** je část života aktivity, kdy je viditelná na obrazovce, ačkoliv nemusí být nutně v popředí (je částečně překryta jinou aktivitou). Tento cyklus začíná metodou onStart() a končí metodou onStop(). Tento cyklus se opakuje s tím, kdy je aktivita viditelná a následně překrytá jinou aktivitou.
- **Životní cyklus aktivity v popředí** se vztahuje především k možnosti uzamčení zařízení a následného znovuspuštění. V tomto případě je, při uspání zařízení, vyvolána metoda onPause() a při následném znovuspuštění se provede metoda onResume().

V následujícím kódu je vidět, které metody aktivity je možné přepsat svými vlastními. Pro bližší informace o funkcích třídy Activity je možno navštívit stránky *http://developer.android.com/reference/android/app/Activity.html*.

```
protected void onCreate(Bundle savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
```

1.5.2 View

Do aktivity je možno kreslit obsah pomocí objektů View. View je visuální prvek, jehož nejvýše hierarchicky postavenými prvky jsou layouty (rodičovské prvky), které určují organizování view potomků, které jsou v layoutu obsaženy. Princip práce s view by se dal přirovnat k principu HTML. Každý visuální prvek View, například tlačítko, musí být obsaženo (být potomkem) nějakého vyššího view. Nejvýše postaveným View, ve kterém jsou standardně obsaženy všechny ostatní View, se nazývá layoutem. Layout by se dal přirovnat vůči HTML například k tagu <*BODY*>. Všechen ostatní obsah visuálních prvků by měl být obsažen v onom nejvýše položeném layoutu. Layoutů je více druhů. Nejběžněji používanými jsou RelativeLayout, LinearLayout a TableLayout. Rozdíl mezi těmito layouty je v podstatě pouze v principu umístění jeho potomků uvnitř jejich těl. Například při umístění tlačítka do nejběžněji používaného RelativeLayoutu je nutné zvolit bod, o který se naše tlačítko „opře“. Můžeme tedy říct, že chceme, aby se tlačítko opřelo o levý a horní okraj nadřazeného layoutu. Nebo můžeme říct, že chceme, aby tlačítko bylo opřené o levý okraj layout a zároveň bylo pod okrajem jiného tlačítka, které již existuje uvnitř layoutu.

Jak bylo již dříve napsáno, pokud není nutno generovat dynamicky se měnící obsah visuálního výstupu aktivity, je lepší využít statického rozvržení layoutu. To je možné pomocí XML layoutu, který se standardně nachází v projektu v adresáři */res/layout* . Tento soubor může být pojmenován alfanumericky a musí mít koncovku *.xml* . K vygenerování XML souboru lze použít WYSIWYG editor, který je obsažen jak v Eclipsu, v případě programování Androida v jazyce Java, tak i ve Visual Studiu/Monodevelop, v případě programování v jazyce C# v Mono for Android. Bohužel pokud začínající programátor nezná principy základního kódového zpracování XML layoutu, tak se může při použití visuálního editoru setkat s mnoha problémy, jejichž řešení mu nebude zřejmé, proto je lepší se nejdříve seznámit se samotným zdrojovým kódem XML [7].

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />
</RelativeLayout>
```

Kód výše je příkladem obsahu XML layoutu, který obsahuje jeden TextView. TextView je obyčejný textový popisek, který je možné měnit a formátovat. Jak je vidět, TextView je umístěn uvnitř RealtiveLayoutu, který svými vlastnostmi šířky android:layout_width určuje, že se má celý layout roztáhnout na celou plochu jeho rodiče, kterým je v našem případě celé okno obrazovky aplikace. Tohoto je dosaženo vlastností match_parrent. To samé je nastaveno pro šířku. Pokud se podíváme na vlastnosti TextView, tak vidíme, že šířka android:layout_width je nastavena na zalamování obsahu. To samé platí i pro výšku. Řádka s android:layout_centerHorizontal="true" určuje, že TextView má být vycentrováno horizontálně. Následuje vycentrování vertikálně. Řádka s textem android:text="@string/hello_word" nastavuje obsah textu pro daný popisek. Zavináč na začátku značí odkaz na uložené hodnoty typu String pod určitým identifikátorem. V našem případě je indentifikátorem hello_word. Za tímto identifikátorem se může skrývat libovolný text, který je uložen uvnitř projektu. Pro začátek je lepší si těchto odkazů zbytečně nevšímat. Toto lze použít později, pokud chceme například udělat vícejazyčnou aplikaci. V samotném XML layoutu není žádný rozdíl, pokud porovnáme layout pro aplikaci v Javě a pro aplikaci v jazyce C#.

1.5.3 Jednoduchá aplikace využívající Activity a View

V následujícím kódu je pro snažší pochopení vytvořen layout aplikace, která obsahuje jedno pole pro zadání textu, tlačítko a textový popisek. Stiskem tlačítka se nastaví obsah popisku na text, který byl zapsán do editovatelného textového pole. Jsou zde uvedeny obě varianty jak pro C# tak pro jazyk Java. Soubor obsahující XML layout je totožný pro oba programovací jazyky.

Pokud chceme vytvořit novou aplikaci, tak je nutné nejdříve v námi zvoleném vývojovém prostředí vytvořit nový projekt. To je možné standardně provést skrze tlačítko vodorovné nabídky „Soubor“ a následně zvolíme možnost založit nový projekt. Poté se nám objeví v nabídce více možností předpřipravených šablon projektů. Nejvhodnější je zvolit „Android Application“. Zvolením této položky vytvoříme univerzální aplikaci, která není závislá na verzi operačního systému Android. Případné omezení verze je následně možné nastavit v souboru AndroidManifest.xml. V případě C# aplikace tvořené pomocí softwaru MS Visual Studio není soubor AndroidManifest vygenerován automaticky, ale k jeho vygenerování dojde až po otevření nastavení projektu po kliknutí pravým tlačítkem na samotný projekt v prohlížeči souborů projektu v samotném Visual Studiu.

Pro vytvoření layoutu neboli rozložení obrazovky si otevřeme layout hlavní aplikace, který bývá vygenerován automaticky se založením projektu. Standardní cesta je v projektu */res/layout*. Po otevření se nám nabízí dvě varianty zobrazení layoutu. Prvním zobrazením je grafický layout, kdy se zobrazí editor s možností úprav kódu tažením prvků pomocí myši. Druhou možností je přímo vyjádření layoutu pomocí kódu. My si nejdříve vygenerujeme kód pomocí visuálního editoru a následně zkontrolujeme podle kódu, který je popsán níže. V grafickém editoru upravíme obsah plochy tak, aby bylo vloženo pole s editovatelným textem tzv. EditText. Dále přidáme tlačítko Button a jako textový popisek použijeme TextView, který může být například Medium Text z nabídky visuálních prvků. Všechny ostatní prvky, které zde mohou být obsaženy od založení projektu jako vzorové, by měly být vymazány. Pokud je přepnuto zobrazení na náhled kódu XML layoutu, tak můžeme spatřit kód, který by se měl podobat kódu popsanému níže.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:ems="10" >

        <requestFocus />
    </EditText>
```

```
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/editText1"
        android:text="Odeslat" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1"
        android:text="Výchozí text"
        android:textAppearance="?android:attr/textAppearanceMedium" />
</RelativeLayout>
```

Ve vygenerovaném zdrojovém kódu XML layoutu můžeme vidět, že celý layout tvoří RelativeLayout, který obsahuje EditText sloužící pro zapsání textu uživatelem. Dále je zde Button tedy tlačítko, které je stisknutelné pro překopírování textu. A nakonec je v layoutu obsažen TextView, které slouží jako popisek, ve kterém měníme text. Pokud si přečteme vlastnosti jednotlivých prvků, tak je vidět, že layout je roztažen na celou obrazovku. Textové pole EditText je pojmenované identifikátorem *editText1*, je zalamované a je zarovnáno doleva a nahoru. Tlačítko je vedeno pod identifikátorem *button1*, je zalamováno a zarovnáno doleva, je umístěno pod textovým polem *editText1* a text tlačítka je nastaven na hodnotu *Odeslat*. Posledním prvkem je popisek TextView, který je veden pod identifikátorem *textView1*, je zalamován na výšku i na šířku, zarovnán vlevo, umístěn pod tlačítkem *button1*. Text je nastaven na hodnotu *Výchozí text* a vzhled je nastaven na velikost medium. Důležité je si uvědomit, že celý soubor XML a jeho jednotlivé prvky vychází z původní třídy View, která byla již popsána výše.

```
package com.example.textboxstlacitkem;

import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.app.Activity;

public class MainActivity extends Activity {
       Button tlacitko;
       EditText textPole;
       TextView popisek;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //vytvoříme si grafické prvky a přiřadíme k nim prvky XML layoutu
        tlacitko = (Button)findViewById(R.id.button1);
        textPole = (EditText)findViewById(R.id.editText1);
        popisek = (TextView)findViewById(R.id.textView1);

        tlacitko.setOnClickListener(new OnClickListener() {

                    public void onClick(View v) {
                          String obsahTextPole = textPole.getText().toString();
                          popisek.setText(obsahTextPole);
                    }
         });
    }
}
```

Na zdrojovém kódu výše můžeme vidět celý obsah hlavní aktivity, která je hlavní vygenerovanou a následně upravenou třídou, jenž je spouštěna po startu aplikace. V třídě jsou nadefinovány položky Button, EditText a TextView. Po spuštění aktivity dojde k vykonání metody onCreate(…). V této metodě jsou jednotlivým vlastnostem třídy (tlačítko atd.) přiřazeny prvky z již načteného layoutu. K přiřazení dochází použitím metody *findViewById(R.id.xxx)*. Namísto trojice *x* je však nutné použít identifikátor, který je nastaven v XML souboru pro jednotlivé visuální prvky. Také je nutné explicitně konvertovat objekt získaný metodou *findViewById(R.id.xxx)* na požadovaný objekt. Po těchto krocích máme vytvořeny instance jednotlivých visuálních prvků, které jsou provázané s XML layoutem. Nyní je potřeba vytvořit metodu, která se provede po stisknutí tlačítka. Toho se dosáhne nastavením třídy *OnClickListener* na dané tlačítko pomocí metody *setOnClickListener(...)*. Uvnitř OnClickListener nastavíme metodu *public void onClick(View v).* Tato metoda je provedena vždy při kliknutí na prvek, kterému byla přiřazena. Uvnitř této metody dojde k získání textu z textového pole a následnému uložení do proměnné *obsahTextPole*. Poté je nastavena získaná hodnota textu popisku a tím dojde k přepsání textu na obrazovce. Samozřejmě nesmí být zapomenuto na importování všech potřebných tříd. V následujícím kódu můžeme vidět, jak vypadá zdrojový kód s totožnými funkcemi napsaný v jazyce C#.

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace TextboxSTlacitkem
```

```
{
    [Activity(Label = "TextboxSTlacitkem", MainLauncher = true, Icon =
"@drawable/icon")]
    public class Activity1 : Activity
    {
        EditText textPole;
        Button tlacitko;
        TextView popisek;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            // nastavíme aktivitě XML layout
            SetContentView(Resource.Layout.Main);

            //vytvoříme si grafické prvky a přiřadíme k nim prvky XML layoutu
            textPole = FindViewById<EditText>(Resource.Id.editText1);
            tlacitko = FindViewById<Button>(Resource.Id.button1);
            popisek = FindViewById<TextView>(Resource.Id.textView1);

            tlacitko.Click += delegate {
                String obsahTextPole = textPole.Text;
                popisek.Text = obsahTextPole;
            };
        }
    }
}
```

Při porovnání obou kódů můžeme vidět, že rozdíly nejsou nijak velké, avšak nejsou ani zanedbatelné. Rozdíly v syntaxi a pojmenování příkazů jsou patrné na první pohled. Na druhý pohled je možné si všimnout, že většina metod v jazyce C# začíná velkými písmeny na rozdíl od metod v jazyce Java. Funkce *FindViewById* v jazyce C# využívá polymorfismu a není tedy nutné explicitně konvertovat objekty. Nejvíce odlišnou částí kódu je vytvoření funkce stisknutí tlačítka. Ta je v jazyce C# vytvořena pomocí *event handleru* a příkazu *delegate*, se ve složených závorkách nachází všechny příkazy, které se mají provést po stisku tlačítka. Také můžeme vidět, že k získání textového obsahu textového pole jsme přistoupili přímo k proměnné typu String. Následně byl nastaven text popisku použitím přímého přístupu k proměnné. Na rozdíl od jazyka Java je při programování v jazyce C# častěji používáno přímo veřejných proměnných. Oproti tomu se při programování Androidu v Javě setkáme téměř vždy s funkcemi, kterými přistupujeme k privátním proměnným.

Pokud si chceme vyzkoušet funkčnost námi vytvořené aplikace, tak je možné provést spuštění aplikace pomocí tlačítka „Run“ v případě vývoje Java aplikace v prostředí Eclipse, nebo pomocí tlačítka zeleného trojúhelníčku v případě aplikace C# v prostředí Visual Studia. Pokud je při tomto spuštění dostupné skutečné zařízení připojené pomocí USB včetně nainstalovaného USB ovladače, tak by měla být aplikace automaticky nainstalována a

spuštěna v daném zařízení. Pokud však zařízení není k dispozici, zobrazí nám prostředí nabídku, kde je možné vybrat běžící emulátor, na kterém má být aplikace spuštěna. Pokud však neběží žádný emulátor, je u této nabídky také možnost spuštění emulátoru. Pokud se po otevření seznamu emulátorů pro spuštění žádný nenachází, tak nejspíše došlo k chybě v průběhu instalace SDK a je vhodné pomocí SDK managera emulátory doinstalovat. Po spuštění a následném vybrání běžícího emulátoru by již měla být vytvořená aplikace spuštěna a zobrazena.

2 Mono for Android

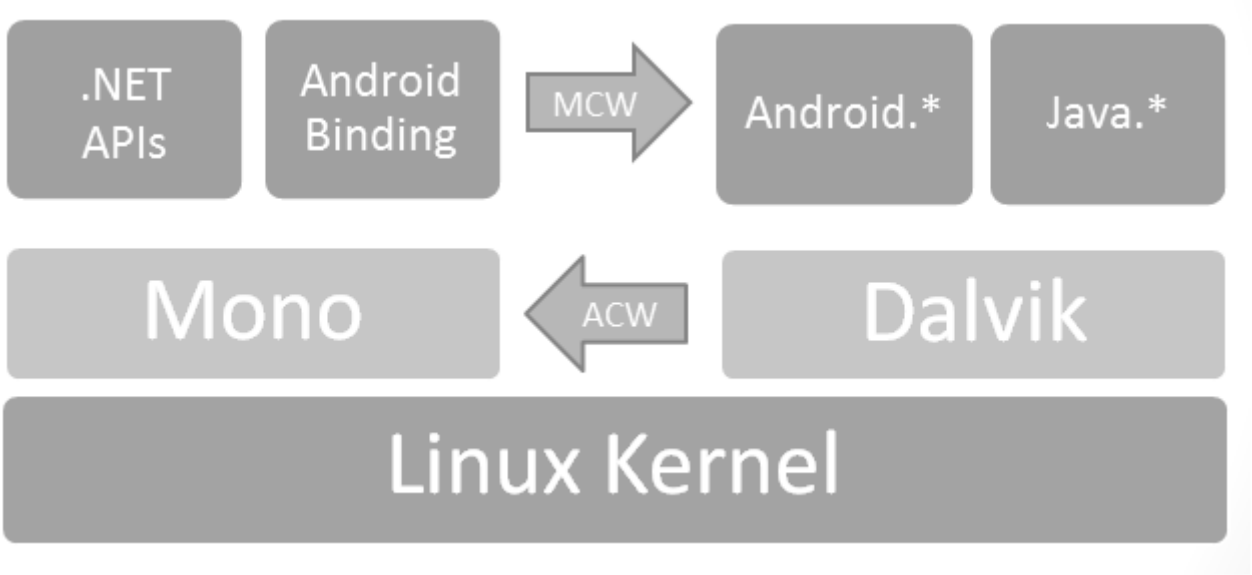
Mono for Android je projekt, který je vyvíjen společností Xamarin. Díky tomuto projektu je možné vyvíjet aplikace pro operační systém Android v prostředí .NET v jazyce C#.

Při programování aplikací pro systém Android je využíváno tzv. API. Tato zkratka je synonymem pro *Application Programming Interface*. Jedná se o sbírku procedur, funkcí, tříd či protokolů dané knihovny. Jedná se tedy o rozhraní, díky kterému se snáze programují aplikace. Díky tomu je možno při programování využívat tohoto rozhraní, které je dodáno samotnou společností Google a.s., která tímto usnadnila práci programátorům aplikací. Důležitou součástí každého API je kvalitní dokumentace. Ta je pro vývoj aplikací v jazyce Java velmi dobrá a podrobná. Avšak dokumentace Mono for Android oproti originální dokumentaci v jazyce Java zaostává a často je možné se setkat s chybějícím zdokumentováním některých méně používaných tříd či metod [8].

2.1 Architektura Mono for Android

Při pohledu na obrázek číslo 2 architektury aplikací pro Android v jazyce C# je vidět, že základní architektura je rozložena do tří vrstev. Nejnižší vrstvou je samotné jádro, které je postaveno na platformě Linux a je napsané v jazyce C. Nad tímto jádrem standardně běží virtual machine Dalvik, který se stará o vykonání Javovského bytekódu aplikace. Mono for Android, pro vykonání kódu napsaného v jazyce C#, využívá pro každou aplikaci instanci svého vlastního virtuálního stroje nazvaného Mono. Ve výsledku tedy běží bok po boku oba virtuální stoje Mono a Dalvik. Oba jsou mezi sebou propojeny a mají svoje vlastní API pro přístup k systémovým prostředkům. Jak Mono tak Dalvik jsou napsány v jazyce C. Aby mohly obě prostředí Mono i Dalvik spolupracovat, používají pro komunikaci mezi sebou

Android Callable Wrappers (ACW) a Managed Callable Wrappers (MCW) v závislosti na směru přenosu dat. Toto propojení zprostředkovávají programy napsané v jazyce assembly, které jsou součástí vygenerované aplikace s příponou .apk .



Obr. 3: Architektura aplikace – přístup k systémovým prostředkům (převzato z [9])

Je nutné si uvědomit, že potřeba komunikace mezi Mono a Dalvik přináší mnoho úskalí. Prvním problémem je, že není možné sdílet proměnné mezi Mono a Dalvik. Vzniká tím potřeba všechny proměnné přenášet, což způsobí vyšší zatížení procesoru a následně neefektivní využití paměti, kdy je nutno mít jednu proměnnou v paměti dvakrát. Tento problém je také umocněn automatickou správou paměti, kterou vlastní jak Dalvik tak Mono. Díky tomu, že nezávisle na sobě běží dva různě garbage collectory, není paměť dostatečně efektivně čištěna a dochází ke zbytečnému držení proměnných v paměti.

Samotné funkce obsluhující například zvukový výstup, grafický výstup a funkce spojené s mobilním přenosem dat neprobíhají v nativním Dalviku, ale je k nim pomocí Dalviku pouze přistupováno. Díky tomu, že Dalvik je jen určitým druhem prostředníka, bylo možné vyvinout samotné Mono, které je do určité míry nezávislé na Dalviku. Ve výsledku je tedy možné při programování v Mono for Android přistupovat k různým funkcím systému Android přímo pomocí .NET API, které usnadňují práci vývojářům zvyklým v tomto prostředí pracovat. Oproti tomu je možné v Mono for Android využít i API z importovaných tříd, které jsou standardní součástí programování Androidu v Javě. Avšak pro jejich vykonání je proveden přenos dat z Mono do Dalviku a následně jsou příkazy provedeny samotným Dalvikem. Pokud je však po aplikaci napsanou v jazyce C# požadován co nejvyšší výkon, tak je vhodné využívat co nejvíce samotné Mono, abychom se vyhnuli přenosu dat mezi prostředími Dalvik

a Mono. V případě, že vyvíjená aplikace není zaměřena na složité výpočty, neměl by činit přenos dat mezi Mono a Dalvik žádné viditelné zpomalení.

2.2 Výkon Mono for Android

Srovnávání výkonosti jazyka Java v porovnání s jazykem C# je mezi programátory často diskutovaným tématem a jednoznačný závěr lze jen těžko vyvodit. Jazyk C# však má výhodu, že při jeho vývoji bylo často použito kódu programovaného na nižší úrovni blíže hardwaru zařízení. Proto by se dalo předpokládat, že by měl být jazyk C# výkonnější, avšak pouze u některých částí kódu. Protože bytekód v jazyce C# neběží na standardním virtuálním stroji, jako desktopové aplikace, ale na virtuálním stroji Mono vyvinutým společností Xamarin, rozhodl jsem otestovat výkon obou programovacích jazyků v praxi. Pro test byly mnou vyvinuty dva jednoduché testovací programy. Každý program pro testování byl vyvinut ve dvou variantách pro oba programovací jazyky s dodržením co nejmenších rozdílů v kódu s ohledem na rozdílné syntaxe obou jazyků. První program byl zaměřen na test výpočetního výkonu, kde jsem předpokládal rozdíly pouze minimální způsobené především kódem jazyka C#, kdy jeho kód by měl být na nižší úrovni blíže hardwaru. Druhý program byl zaměřen na test výpočetní rychlosti při použití grafických prostředků systému Android. V druhém testu byly očekávány již větší rozdíly a to především na základě prohlášení technického ředitele společnosti Xamarin, který tvrdil, že Mono je schopné efektivněji využívat grafické prostředky systému Android [10].

2.2.1 Aplikace testující výpočetní výkon jazyků C# a Java

V projektu, ve kterém jsem se zabýval vývojem Android aplikace pro objednávací terminál jídel v jídelně, který souvisí s touto bakalářkou prací, bylo potřeba načíst velké množství strávníků ze souboru do paměti. Při této práci bylo nutné načítat od jednoho až do pěti tisíc strávníků včetně jejich osobních dat do paměti. Každý strávník mohl mít zhruba deset položek, které bylo potřeba uložit v paměti pro pozdější rychlé načítání. Většina dat byla ukládána v objektech typu *String*. V průběhu práce bylo potřeba neustále kontrolovat správnost všech dat, aby nedošlo k pozdějšímu pádu aplikace pro nesprávně uložená data. Při spuštění této aplikace však docházelo k nepříjemně dlouhému načítání, které při pěti tisících strávnících dosahoval času až dvou minut. Tato prodleva byla způsobena složitými operacemi s textovými řetězci. Proto jsem se rozhodl test výpočetního výkonu orientovat na operace s řetězci, kdy operace s několika tisíci textovými řetězci by měla být dostatečně zatěžující.

Pro test výkonu aplikace napsané v jazyce C# v porovnání s aplikací v jazyce Java byla zvolena práce s řetězci a jejich ukládání do kolekce. Tímto testem bylo zamýšleno prověřit rozdíly ve výkonu virtuálního stroje Mono a Dalvik v situaci, která částečně simuluje práci s velkým množstvím dat v podobě řetězců a jejich ukládáním do kolekce. Tento test byl zvolen, protože rozdíly ve výkonu jsou postřehnutelné pouze při časově náročnějších operacích a tím práce s velkým množstvím dat v aplikacích pracující s databází jednoznačně je. Dalším důležitým předpokladem pro získání co nejpřesnějších dat byla nutnost co nejvíce se vyhnout přenášení dat mezi virtuálními stroji Mono a Dalvik. V nativní aplikaci v jazyce Java tento problém nebylo nutné pro nepřítomnost Mono řešit. Avšak u aplikace napsané v jazyce C# bylo nutné použít třídy, které vykonává Dalvik, pouze pokud nebyla jiná možnost. Ve výsledku tedy pro veškerou práci s řetězci, kolekcí i měřením času byly využity pouze třídy obsažené v .NET. Při každém testu byl nejdříve proveden totožný test výkonu, kde však nebyl měřen čas vykonání kódu. Tento test sloužil k inicializaci všech proměnných, aby nedošlo k propadu výkonu, způsobeném prací s pamětí.

Níže je vidět výřez zdrojového kódu v jazyce C#. Jedná se o cyklus, jehož provedení bylo měřeno. Výřez kódu v jazyce Java byl téměř totožný pouze s rozdílem, kdy jsou v jazyce Java zapisovány názvy metod malými písmeny. Kompletní zdrojový kód aktivit testovacích programů v jazyce C# i Java je možno nalézt v příloze A1 a A2.

Cyklus nejdříve generuje řetězec, který se každým průchodem liší o číslice na konci. Tím je také měněna jeho délka. Vzniklý řetězec je následně oříznut o bílé znaky. Konkrétně je odstraněna mezera na začátku. Poté je testováno, zda je v řetězci obsažen řetězec *text*, avšak výsledek není zaznamenáván. Nakonec je výsledný řetězec přidán do kolekce a proměnná s původním řetězcem je uvolněna. Tento test byl prováděn pro 10 000 průchodů cyklem a následně pro 100 000 průchodů, kdy byla měřena doba potřebná pro provedení všech cyklů. Měření bylo provedeno na dvou rozlišných mobilních telefonních zařízeních s operačním systémem Android. Každé měření bylo opakováno pětkrát a jako výsledek byl brán aritmetický průměr jednotlivých měření. Naměřené výsledky je možno vidět v tabulce číslo 1.

```
for (int i = 0; i < pocetCyklu; i++)
{
       generovanyString = " text" + i;   //generujem řetězec
       generovanyString = generovanyString.Trim();     //odstraníme mezeru na začátku
```

```
        generovanyString.Contains("text");//otestujeme, zda je obsažen v řetězci "text"

        kolekceStringu.Add(generovanyString); //přidame řetězec do kolekce
        generovanyString = null;   //uvolníme proměnnou
}
```

Tab.č. 1: Výsledky měření výpočetního času algoritmu

	SE Xperia Mini X10 pro				Samsung Galaxy Ace			
	C#		Java		C#		Java	
počet cyklů	10 000	100 000	10 000	100 000	10 000	100 000	10 000	100 000
1.čas[ms]	466	3975	777	7589	213	2199	399	3492
2.čas[ms]	400	4756	813	7505	300	2264	383	3269
3.čas[ms]	430	3963	831	7453	204	2189	453	3419
4.čas[ms]	356	3989	835	7657	249	2227	425	3365
5.čas[ms]	429	4840	761	7391	201	3560	360	3362
aritmetický průměr[ms]	**416**	**4305**	**803**	**7519**	**233**	**2488**	**404**	**3381**

Z výsledků je patrné že Mono for Android zcela jasně poráží nativní aplikaci v Javě. Z výsledků plyne, že aplikace napsaná v jazyce C# potřebuje pro operace s textovými řetězci a kolekcí zhruba polovinu výpočetního času oproti aplikaci v jazyce Java. Tento výsledek dokazuje, že virtuální stroj Mono využívá výkonu daného hardwaru výrazněji lépe než nativní Dalvik.

2.2.2 Aplikace testující grafický výkon jazyků C# a Java

Pro test grafického výkonu jsem se rozhodl testovat dvojrozměrnou grafiku. Pro vykreslení 2D grafického výstupu je možno použít klasickou Java třídu Canvas, která se chová jako určitý druh kreslícího plátna, na které můžeme kreslit obrazce, základní tvary a jejich kombinováním můžeme vytvářet složitější obrazce. Také je možno použít velmi populární OpenGL knihovnu pro grafický výstup, avšak tato knihovna je primárně určena pro komplikované hry, či trojrozměrné aplikace a její použití na jiný druh aplikace se mi jeví jako zbytečně komplikovaný krok. Proto mě v tomto testu zajímá, zda je možné pozorovat rozdíly ve výkonu aplikace napsané v jazyce C# v porovnání s jazykem Java při využití třídy Canvas.

Vzhledem k tomu, že třída Canvas se nachází v API nativního Dalviku předpokládá se, že rozdíl ve výkonu by měl být minimální, protože virtuální stroj Mono by měl předat instrukce pro vykreslení Dalviku. Teoreticky by bylo možné předpokládat dokonce nižší výkon u aplikace napsané v jazyce C# pro její nutnost komunikace mezi virtuálními stroji.

Aplikace, která byla vytvořena pro testování grafického výkonu za použití třídy Canvas, byla navržena pro běh ve dvou vláknech. První vlákno je standardní vlákno, které se spustí s metodou onCreate() každé výchozí aktivity. Toto vlákno bylo přítomno ve všech předcházejících aplikacích. Nyní je však vytvořeno také vlákno, které se stará o vykreslování grafického výstupu do Canvas. Pro snažší orientaci v kódu byla aplikace sestavena bez použití XML souboru s layoutem, ale layout byl vytvořen přímo v kódu a následně do něj byly umístěny všechny nezbytné grafické prvky. V tomto příkladu je tedy také možno vidět, jak lze bez použití XML souboru vytvořit layout uvnitř programu.

Test probíhá na základě vykreslení úsečky tak, že jeden konec úsečky je stále na stejném místě, například uprostřed obrazovky, a druhý konec úsečky mění svoji pozici na základě goniometrických funkcí sinus a cosinus. Pomocí sinu a cosinu jsou dopočítávány souřadnice x a y druhého konce úsečky. Ve výsledku úsečka rotuje kolem svého prvního krajního bodu a svým pohybem vyplňuje barvou obsah kruhu. V tomto testu je měřena doba, za kterou je schopna přímka opsat pětkrát celý kruh. Celá rovina 2Π je rozdělena na 240 dílů a díky tomu před jedním opsáním kruhu musí dojít ke 240 vykreslení úsečky. Výsledná doba je měřena v milisekundách a za předpokladu znalosti počtu opsaných kruhů a počtu kroků (vykreslení) pro opsání jednoho kruhu, je možné dopočítat množství snímků (překreslení) za jednu sekundu. Vypočítané množství překreslení za sekundu je předpokladem výsledného porovnání výkonu. Čím vyšší je hodnota překreslení za sekundu, tím je vyšší výkon aplikace. Všechna měření byla prováděna pětkrát a jako směrodatný výsledek byl uvažován aritmetický průměr všech pěti měření s následným zaokrouhlením.

Tab. č. 2: Výsledky měření grafického výkonu ve snímcích za sekundu

	SE Xperia Mini X10 pro		Samsung Galaxy Ace	
	C#	Java	C#	Java
1. měření [fps]	82	81	87	85
2. měření [fps]	83	82	89	70
3. měření [fps]	81	83	90	76
4. měření [fps]	84	84	87	89
5. měření [fps]	84	83	90	88
aritmetický průměr [fps]	**83**	**83**	**89**	**82**

Jak je možno vidět v tabulce číslo 2, výsledný výkon v jednotkách snímků za sekundu, je téměř totožný pro oba programovací jazyky. U jednoho ze dvou testovaných zařízení se jeví

aplikace napsaná v jazyce C# o téměř 10% rychlejší, avšak takto malý rozdíl mohl být způsoben i konkrétním stavem využité paměti a zatížení zařízení. Další negativní vliv na výsledek může být způsoben odlišnou konstrukcí aplikace vycházející z rozdílné syntaxe jazyků. Ve výsledku jsem však přesvědčen, že je možné tvrdit, že aplikace napsaná v prostřední Mono for Android je schopná vykreslovat grafický výstup za použití třídy Canvas přinejmenším stejně rychle jako aplikace napsaná v jazyce Java. Oba kompletní zdrojové kódy v jazycích C# a Java aktivity, použité k testování grafického výkonu, je možné najít v příloze B1 a B2.

2.3 Omezení Mono for Android

Prvním omezením při tvorbě aplikací v Mono pro Android je nutnost přibalení samotného virtuálního stroje včetně pomocných programů v jazyce assembly do balíku *.apk* pro instalaci aplikace na zařízení. Ve výsledku je možné balík, který by v nativní Javě měl velikost půl megabytu, vygenerovat ve velikost třeba i pět megabytů. Tato skutečnost může znepříjemnit stahování aplikace uživatelů připojeným pomocí datově omezených mobilních sítí. S potřebou přibalení virtuálního stroje a programů pro provázání virtuálních strojů souvisí i nutnost vygenerovat část byte kódu až přímo na cílovém zařízení, čímž trpí především rychlost nabíhání aplikací psaných pomocí Mono for Android.

Ačkoliv Mono for Android obsahuje některá technická omezení při psaní zdrojového kódu, v porovnání s nativním jazykem Java je zásadním omezením malá komunita programátorů. Při tvorbě aplikací v Mono se programátor často setká s problémem, kdy musí nahlédnout do rad a návodů, které se vyskytují napsané pouze v jazyce Java. Následná realizace informací získaných ze zdrojových kódů v jazyce Java a jejich přepsání do jazyka C# může být velmi frustrující. Například třída, která je importována do projektu Mono na základě třídy obsažené v API nativní Javy, může být ochuzena o metodu, která je potřebná pro realizaci aplikace. Tento problém většinou způsobí nutnost hledání náhrady v celém velmi obsáhlém API Mono for Android a doba potřebná pro realizaci kódu se může až několikanásobně prodloužit.

2.4 Vydání nové verze Xamarin 2.0

V průběhu zpracování této práce byla společností Xamarin na trh uvedena druhá verze vývojového prostředí Xamarin 2.0. Tato verze přináší mnoho vylepšení a změn. První z těchto

změn je zánik původního vývojového prostředí Monodevelop, které bylo nahrazeno novým prostředím Xamarin Studio. Tato změna je doprovázena lepším visuálním rozložením ovládacích prvků prostředí s usnadněním vývoje aplikací pro více platforem najednou.

Další novinkou v nové verzi Xamarin 2.0 je možnost nákupu předpřipravených aplikací přímo prostřednictvím vývojového prostředí. Nákup je možno uskutečnit prostřednictvím služby nazvané Xamarin Component Store, což je služba zamýšlená k usnadnění práce vývojářů nutné pro vytvoření aplikací. Již v původní verzi byla obsažena podpora vývoje Mono for Android aplikací v prostředí MS Visual Studio, avšak v nové verzi přibyla podpora vývoje aplikací v prostředí MS Visual Studio pro operační systém iOs, čímž dostála společnost Xamarin svým slibům ohledně usnadnění vývoje aplikace pro více platforem najednou. Samozřejmostí v nové verzi je obsažení podpory nejnovějších verzí mobilních operačních systémů, která je pro celý projekt Mono zásadní vlastností pro jeho praktickou použitelnost [11].

Naprostá většina vylepšení nové verze se dá považovat za úspěch, avšak současně došlo k výraznému zdražení, kdy původní komerční verzi bylo možné zakoupit za 399$. Součástí této licence byla možnost instalace softwaru na třech pracovních stanicích. V současné době je však možné zakoupit komerční verzi za 999$. Tato licence se váže právě na jednu zvolenou platformu a právě na jednu osobu. Současně přibyla i levnější varianta nákupu licence pro účely individuálního vývoje indie aplikací, kterou je možno pořídit levněji a to za 299$.V této verzi však chybí například podpora MS Visual Studia.

3 Vzorová aplikace pro zápis do souboru na SD kartu

Na závěr této práce bych rád demonstroval jeden z velmi často řešených problémů, který se zabývá zápisem dat do textového souboru. Pokud je potřeba přibalit do aplikace například obrázky, které jsou součástí visuálního vzhledu aplikace, je vhodné je přidat přímo do projektu aplikace. Pro takovéto soubory jsou v projektu připraveny složky, které byly pro tento účel vytvořeny. Například složka *assets* je vhodná pro uložení vlastních fontů s příponou .ttf, složka *res* obsahuje mimo jiné složku *drawable*, která je vhodná pro uložení vlastních obrázků včetně ikony aplikace. Avšak existují i případy, kdy je potřeba data číst nebo ukládat přímo na SD kartu. Může se jednat například o prohlížeč fotografií vytvořených vestavěným fotoaparátem nebo přehrávač hudby uložené na SD kartě.

V tomto případě je níže popsána aplikace, která ukládá data do souboru a následně je ze souboru čte. Pro větší komplexnost aplikace, jsem sestavil zdrojový kód, který ukládá text do souboru v námi zvoleném kódování textu, což je především kvůli kompatibilitě souborů nutné důsledně dodržovat. Rozvržení aplikace tvoří jeden *EditText*, dvě tlačítka *Button* a jeden *TextView*. Pokud je třeba zapsat námi zvolený text do souboru, tak je nejdříve nutné tento text zapsat do pole *editovatelnyText*, následně pomocí prvního tlačítka *tlacitkoZapsat* dojde k zapsání textu do souboru a ke smazání textu v poli *editovatelnyText.* Pokud soubor, který byl zvolen pro ukládání, na SD kartě neexistuje, je automaticky vytvořen. V opačném případě je námi zvolený text přidán na konec souboru. Ve chvíli, kdy na SD kartě existuje textový soubor s námi vepsaným obsahem, je možno zmáčknout druhé tlačítko *tlacitkoVypsat.* Tím dojde k přečtení souboru a jeho následném smazání. Přečtený obsah je vrácen funkcí jako řetězec a pokud není roven *null,* tak dojde k jeho vypsání v TextView s názvem *textovyPopisek*. Níže je možno vidět zdrojový kód aplikace napsané v jazyce C# v prostředí Mono for Android. Pro porovnání s identickou aplikací v jazyce Java je možno nahlédnout do přílohy C této práce.

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Java.Nio.Charset;
using System.IO;

namespace CteniZapisSoubor
{
  [Activity(Label = "CteniZapisSoubor", MainLauncher = true, Icon = "@drawable/icon")]
  public class Activity1 : Activity
  {
    //Deklarace grafických prvků
    EditText editovatelnyText;
    Button tlacitkoZapsat;
    Button tlacitkoVypsat;
    TextView textovyPopisek;

    protected override void OnCreate(Bundle bundle)
    {
      base.OnCreate(bundle);
      SetContentView(Resource.Layout.Main);

      //inicializace grafických prvků z layoutu
      editovatelnyText = (EditText)FindViewById(Resource.Id.editText1);
      tlacitkoZapsat = (Button)FindViewById(Resource.Id.button1);
      tlacitkoVypsat = (Button)FindViewById(Resource.Id.button2);
      textovyPopisek = (TextView)FindViewById(Resource.Id.textView1);

      tlacitkoZapsat.Click += delegate //příkazy po stisknutí tlačítka-zapsat
      {
```

```
      String textKZapsani = editovatelnyText.Text;

      if (!textKZapsani.Equals(""))
      {
        if (zapsatDoSouboru(textKZapsani) == false)
          vypisVarovani("Zápis do souboru na SD kartu se nezdařil!");
        else
        {
          vypisVarovani("Zápis do souboru na SD kartu byl proveden.");
          editovatelnyText.Text = "";
        }
      }
    };

    tlacitkoVypsat.Click += delegate //příkazy po stisknutí tlačítka-vypsat
    {
      String textSouboru = precistSoubor();

      if (textSouboru != null)
        textovyPopisek.Text = textSouboru;
      else
        vypisVarovani("Čtení souboru se nezdařilo");
    };
  }

  //metoda pro výpis upozornění v případě problémů
  public void vypisVarovani(String varovani)
  {
    Toast.MakeText(this, varovani, ToastLength.Short).Show();
  }

  //metoda pro zápis do souboru, vrací bool, zda se zápis zdařil
  public bool zapsatDoSouboru(String str){
    String cesta = Android.OS.Environment.ExternalStorageDirectory.ToString() +
                                                          "/souborData.txt";
    FileStream fStream = null;
    StreamWriter rWrite = null;

    try
    {
      if (File.Exists(cesta))
      {
        fStream = new FileStream(cesta, FileMode.Append, FileAccess.Write);
        rWrite = new StreamWriter(fStream,
                              System.Text.Encoding.GetEncoding("windows-1250"));
        rWrite.Write(str + "\n");
        rWrite.Close();
        return true;
      }
      else
      {
        fStream = new FileStream(cesta, FileMode.Create, FileAccess.Write);
        rWrite = new StreamWriter(fStream,
                               System.Text.Encoding.GetEncoding("windows-1250"));
        rWrite.Write(str + "\n");
        rWrite.Close();
        return true;
      }
    }
    catch(Exception ex)
    {
      return false;
    }
```

```
    }

    //metoda pro čtení ze souboru, vrací data jako řetězec
    public String precistSoubor()
    {
      String cesta = Android.OS.Environment.ExternalStorageDirectory.ToString() +
                                                                    "/souborData.txt";
      String obsahSouboru = null;
      FileStream fStream = null;
      StreamReader rRead = null;

      try
      {
        fStream = new FileStream(cesta, FileMode.Open, FileAccess.Read,
                                                                      FileShare.Read);
        rRead = new StreamReader(fStream,
                                 System.Text.Encoding.GetEncoding("windows-1250"));
        obsahSouboru = rRead.ReadToEnd();
        File.Delete(cesta);
      }
      catch (Exception ex)
      {
        return null;
      }
      finally
      {
        if(rRead!=null)
        rRead.Close();
        if(fStream!=null)
          fStream.Close();
      }
      return obsahSouboru;
    }
  }
}
```

Pro spuštění výše popsané aplikace je však ještě nutné zadat oprávnění pro čtění a zápis na SD kartu do souboru *AndroidManifest*. Tento soubor je v Javě generován automaticky s vytvořením projektu, avšak při použití Mono for Android nemusí být tento soubor automaticky vygenerován. Pro ruční vygenerování je nutné klepnout pravým tlačítkem myši na celou složku projektu v MS Visual Studio a následně vybrat položku *Properties*. V zobrazené nabídce stačí vybrat položku *Android Manifest* a následně se vám nabídne možnost vygenerování souboru. Poté je nutné vložit oprávnění do vnitřku tagu <*manifest*>. Oprávnění je možné vložit v následujícím tvaru.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Závěr

Ačkoliv je těžké si představit množství práce, které stojí za celým projektem portu jazyka C# pro operační systém Android. Mono for Android společnosti Xamarin, je produktem, který splnil většinu mých očekávání. Vzhledem k bohatství neustále se rozrůstajících a měnících knihoven jazyka Java pro systém Android, je jejich téměř kompletní port do jazyka C# obdivuhodným výkonem. Navzdory nedávnému výraznému zdražení verze pro komerční využití, není dle mého názoru tato cena neopodstatněná. Otázkou však stále zůstává, co nám samotné Mono na operačním systému Android může přinést za výhody. Slibovaný nárůst grafického výkonu se v provedeném testu dvourozměrné grafiky nepotvrdil, avšak časová náročnost pro zpracování řetězců dosahovala téměř poloviční doby v porovnání s jazykem Java. Tím bylo dokázáno, že architektura jazyka C# má blíže k hardwarové části zařízení než jazyk Java, čímž umožňuje efektivněji využít výkonu daného zařízení. Tato výhoda však může být částečně zpochybněna možností vložit do aplikace pro systém Android kód napsaný v jazyce C. Takto vložený kód v jazyce C má potenciál pro vyšší výkon, než je tomu v případě jazyka C#. Avšak tento postup je doporučován pouze pro složité matematické operace. Aplikace napsané v jazyce C# májí oproti jazyku Java výhodu vyššího výkonu. Při dodržení určitých zásad psaní aplikace je možné z jednoho zdrojového kódu bez složitějších úprav vytvořit aplikaci pro platformu Android, iOs i pro platformu Windows Phone. Dalším zřejmým důvodem pro pořízení si tohoto produktu je předchozí zkušenost s jazykem C#, kdy by bylo učení se nového jazyka časově náročné. Tento důvod pro pořízení je však zavádějící, protože pokud budeme tvořit reálné aplikace, bude nutno nahlížet na zdrojové kódy v jazyce Java, kde je komunita vývojářů podstatně bohatší o rady a návody. Nehledě na to, že rozdíly mezi jazyky C# a Java nejsou nikterak zásadní. Ve výsledku si myslím, že Mono for Android má smysl především pro vývoj aplikací pro více platforem najednou, kdy finanční prostředky ušetřené pro odstranění nutnosti vícenásobného vývoje aplikace mohou velmi snadno přesáhnout náklady spojené s pořízením produktu Mono společnosti Xamarin.

Seznam literatury a informačních zdrojů

[1] PAGE, Larry. Official Blog: Update from the CEO. Google: Official Blog [online]. 2013 [cit. 2013-04-17]. Dostupné z: http://googleblog.blogspot.cz/2013/03/update-from-ceo.html

[2] Android (operační systém). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-28]. Dostupné z: http://cs.wikipedia.org/wiki/Android_%28opera%C4%8Dn%C3%AD_syst%C3%A9m%29

[3] PITNER, Tomáš. *Java - začínáme programovat*. Praha: Grada Publishing, spol. s.r.o., 2002, s. 14. ISBN 80-247-0295-9.

[4] JOSH, Lowensohn. Jury clears Google of infringing on Oracle patents. *ZDNet* [online]. 2012 [cit. 2013-04-17]. Dostupné z: http://www.zdnet.com/blog/btl/jury-clears-google-of-infringing-on-oracle-patents/77897

[5] App Framework: Android Architecture. In: *Android Developers* [online]. 2013 [cit. 2013-04-17]. Dostupné z: http://developer.android.com/about/versions/index.html

[6] Activity. *Android Developers* [online]. 2013 [cit. 2013-05-18]. Dostupné z: http://developer.android.com/reference/android/app/Activity.html

[7] Vyvíjíme pro Android: První krůčky. KONEČNÝ, Matěj. *Zdroják.cz* [online]. 2012 [cit. 2013-05-19]. Dostupné z: http://www.zdrojak.cz/clanky/vyvijime-pro-android-prvni-krucky/

[8] API. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-22]. Dostupné z: http://cs.wikipedia.org/wiki/API

[9] Architecture | xamarin. XAMARIN. *Xamarin* [online]. 2013 [cit. 2013-05-22]. Dostupné z: http://docs.xamarin.com/guides/android/advanced_topics/architecture

[10] Android bez Javy? Podle společnosti Xamarin to není problém. PELECH, Tadeáš. *SmartWorld.cz* [online]. 2012 [cit. 2013-05-24]. Dostupné z: http://smartworld.cz/android/android-bez-javy-podle-spolecnosti-xamarin-to-neni-problem-2836

[11] Announcing Xamarin 2.0. *Xamarin* [online]. 2013 [cit. 2013-05-28]. Dostupné z: http://blog.xamarin.com/announcing-xamarin-2.0/

Přílohy

Příloha A1 – Zdrojový kód aktivity - Výpočetní Benchmark – C#

```
using System;

using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using System.Collections.Generic;
using System.Diagnostics;

namespace BenchmarkVypocetni
{
    [Activity(Label="BenchmarkVypocetni", MainLauncher=true, Icon="@drawable/icon")]
    public class Activity1 : Activity
    {
      long vysledek;
      String generovanyString;

      Button button1;
      Button button2;
      TextView textView1;

      Stopwatch stopky = new Stopwatch();

      List<String> kolekceStringu = new List<String>();

      protected override void OnCreate(Bundle bundle)
      {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);

        button1 = FindViewById<Button>(Resource.Id.button1);
        button2 = FindViewById<Button>(Resource.Id.button2);
        textView1 = FindViewById<TextView>(Resource.Id.textView1);

        button1.Click += delegate
        {
          testVykonu(10000); //slouží ke správní inicializaci proměnných, brání
                             //zkreslení výsledku
          vysledek = testVykonu(10000);
          textView1.Text=(textView1.Text + "\ntest 10 000 cyklů: " + vysledek + "ms");
        };

        button2.Click += delegate
        {
          testVykonu(100000); //slouží ke správní inicializaci proměnných, brání
                              //zkreslení výsledku
          vysledek = testVykonu(100000);
          textView1.Text = (textView1.Text+"\ntest 100 000 cyklů: "+vysledek+"ms");
        };
      }

      //Metoda pro testování výpočetní náročnosti práce s textovými řetězci
      public long testVykonu(int pocetCyklu)
      {
        stopky.Reset();
        stopky.Start();
```

```
        for (int i = 0; i < pocetCyklu; i++)
        {
          generovanyString = " text" + i; //generujem řetězec s mezerou na začátku a
                                          //variabilním číslem na konci
          generovanyString = generovanyString.Trim(); //odstraníme mezeru na začátku

          generovanyString.Contains("text"); //otestujeme zda je obsažen "text"

          kolekceStringu.Add(generovanyString); //přidáme řetězec do kolekce
          generovanyString = null;       //uvolníme proměnnou
        }

        stopky.Stop();

        kolekceStringu.Clear();

        return stopky.ElapsedMilliseconds;
      }
    }
}
```

Příloha A2 – Zdrojový kód aktivity - Výpočetní Benchmark – Java

```
package com.example.benchmarkvypocetni;

import java.util.ArrayList;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
  long startTestu;
  long konecTestu;
  long vysledek;
  String generovanyString;

  Button button1;
  Button button2;
  TextView textView1;

  ArrayList<String> kolekceStringu = new ArrayList<String>();

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    button1 = (Button)findViewById(R.id.button1);
    button2 = (Button)findViewById(R.id.button2);
    textView1 = (TextView)findViewById(R.id.textView1);

    //nastaví reakci na stisknuté tlačítko s 10 000 cykly
    button1.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        testVykonu(10000); //slouží ke správní inicializaci proměnných, brání
```

```
                        //zkreslení výsledku
        vysledek = testVykonu(10000);
        textView1.setText(textView1.getText()+"\ntest 10 000 cyklů: "
                        +vysledek+"ms");
      }
    });

    //nastaví reakci na stisknuté tlačítko s 100 000 cykly
    button2.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        testVykonu(100000); //slouží ke správní inicializaci proměnných, brání
                        //zkreslení výsledku
        vysledek = testVykonu(100000);
        textView1.setText(textView1.getText()+"\ntest 100 000 cyklů: "
                        +vysledek+"ms");
      }
    });
  }

  //Metoda pro testování výpočetní náročnosti práce s textovými řetězci
  public long testVykonu(int pocetCyklu){
    startTestu = System.currentTimeMillis();

    for(int i = 0; i<pocetCyklu; i++){
      generovanyString = " text"+i; //generujem řetězec s mezerou na začátku a
                                    //variabilním číslem na konci
      generovanyString = generovanyString.trim(); //odstraníme mezeru na začátku

      generovanyString.contains("text"); //otestujeme zda je obsažen "text"

      kolekceStringu.add(generovanyString); //přidáme řetězec do kolekce
      generovanyString = null;  //uvolníme proměnnou
    }

    konecTestu = System.currentTimeMillis();

    kolekceStringu.clear(); //odstraníme data z kolekce

    return konecTestu-startTestu;
  }
}
```

Příloha B1 – Zdrojový kód aktivity - Grafický Benchmark – C#

```
using System;

using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Android.Graphics;
using System.Threading;
using System.Diagnostics;
using Android.Util;
using Java.Lang;

namespace BenchmarkGraficky
{
```

```
  [Activity(Label="BenchmarkGraficky", MainLauncher=true, Icon="@drawable/icon")]
  public class Activity1 : Activity
  {
    //Realizace visuálnich prvků
    public RelativeLayout relativeLayout;
    public MojeView mojeView;
    public Button spustTlacitko;
    public TextView vysledkyText;

    public bool testBezi = false;
    public long pocetProvedenychOtocek = 0;
    public Stopwatch stopky;
    public long vysledek;
    public int fps;

    protected override void OnCreate(Bundle bundle)
    {
      base.OnCreate(bundle);

      RelativeLayout.LayoutParams parametryNahledu = new
RelativeLayout.LayoutParams(Android.Widget.RelativeLayout.LayoutParams.WrapContent,
Android.Widget.RelativeLayout.LayoutParams.WrapContent);

      parametryNahledu.SetMargins(0, 80, 0, 0);

      relativeLayout = new RelativeLayout(this);
      mojeView = new MojeView(this,this);
      spustTlacitko = new Button(this);
      vysledkyText = new TextView(this);

      vysledkyText.Text = "Výsledky měření:";
      spustTlacitko.Text = "Spustit test";

      stopky = new Stopwatch();

      spustTlacitko.Click += delegate
      {
        if(testBezi==false)
        {
          stopky.Start();
          testBezi=true;
        }
        else{
          VypisVarovani();
        }
      };

      relativeLayout.AddView(mojeView);
      relativeLayout.AddView(spustTlacitko);
      relativeLayout.AddView(vysledkyText,parametryNahledu);

      SetContentView(relativeLayout);  // nastaví náhled obrazovky na relativeLayout
    }

    protected override void OnPause()
    {
      base.OnPause();
      mojeView.pause();
    }

    protected override void OnResume()
    {
```

```
      base.OnResume();
      mojeView.resume();
    }

    public void VypisVarovani()
    {
      Toast toast = Toast.MakeText(this, "Test již probíhá !", ToastLength.Long);
      toast.Show();
    }

  }
  public class MojeView : SurfaceView
  {
    Activity1 odkazActivity1;

    System.Threading.Thread t = null;
    ISurfaceHolder holder;
    bool isRunning = false;

    public MojeView(Context context, Activity1 act1) : base(context)
    {
      this.odkazActivity1 = act1;
      holder = Holder;
    }

    public void Run()
    {
      Paint paint = new Paint();
      paint.Color = Color.Red;
      paint.StrokeWidth = 2;

      Canvas c;
      double uhel = 0;

      int polomer = 30;
      int krokuVKruhu = 240; //počet kroků na vykreslení jednoho kruhu
      double krokUhlu = (2 * System.Math.PI) / (double)krokuVKruhu;
      int maxOtocek = 5;

      while (isRunning == true)
      {
        if (!holder.Surface.IsValid)
        {
          continue; //při neplatném Surface nedojde ke kreslení
        }

        c = holder.LockCanvas();

        if (odkazActivity1.testBezi == true)
        {
          c.DrawLine(200, 40, 200 + FloatMath.Sin((float)uhel) * polomer, 40 +
                      FloatMath.Cos((float)uhel) * polomer, paint);

          uhel += krokUhlu;
          if (uhel > 2 * System.Math.PI)
          {
            uhel = 0;
            if (paint.Color == Color.Red)
              paint.Color = Color.Blue;
            else
              paint.Color = Color.Red;
```

```
            odkazActivity1.pocetProvedenychOtocek++;

            if (odkazActivity1.pocetProvedenychOtocek == maxOtocek)
            {
              odkazActivity1.stopky.Stop();
              odkazActivity1.testBezi = false;
              odkazActivity1.pocetProvedenychOtocek = 0;

              odkazActivity1.vysledek = odkazActivity1.stopky.ElapsedMilliseconds;
              odkazActivity1.stopky.Reset();
              odkazActivity1.fps = (int)System.Math.Round((double)1000 /
        ((double)odkazActivity1.vysledek / (double)maxOtocek / (double)krokuVKruhu));
                                                  //vypočte počet snímků za sekundu

              odkazActivity1.RunOnUiThread(delegate //změna zobrazení-nutně v UIvlákně
              {
                odkazActivity1.vysledkyText.Text = odkazActivity1.vysledkyText.Text +
                 "\n" + odkazActivity1.vysledek + "ms, " + odkazActivity1.fps + "fps";
              });
            }
          }
        }

        holder.UnlockCanvasAndPost(c);
      }
    }

    // metoda, která chrání vlákno kreslení před pozastavením zařízení
    public void pause()
    {
      isRunning = false;
      while (true)
      {
        try
        {
          t.Join();
        }
        catch (InterruptedException e)
        {
          e.PrintStackTrace();
        }
        break;
      }
      t = null;
    }

    // metoda, která spustí vlákno kreslení po opětovné spuštění zařízení
    public void resume()
    {
      isRunning = true;
      t = new System.Threading.Thread(new ThreadStart(Run));
      t.Start();
    }
  }
}
```

Příloha B2 – Zdrojový kód aktivity - Grafický Benchmark – Java

```
package com.example.benchmarkkruh;

import android.os.Bundle;
import android.app.ActionBar.LayoutParams;
```

```
import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.FloatMath;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
  //deklarace visuálnich prvků
  RelativeLayout relativeLayout;
  MojeView mojeView;
  Button spustTlacitko;
  TextView vysledkyText;

  boolean testBezi = false;
  long pocetProvedenychOtocek = 0;
  long casStartTestu;
  long casKonecTestu;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //LayoutParams slouží k nastavení umístění visuálního prvku uvnitř layoutu
    RelativeLayout.LayoutParams parametryNahledu = new
RelativeLayout.LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        parametryNahledu.setMargins(0, 80, 0, 0);

    //inicializace visuálních prvků
    relativeLayout = new RelativeLayout(this);
    mojeView = new MojeView(this);
    spustTlacitko = new Button(this);
    vysledkyText = new TextView(this);

    //mojeView.setBackgroundColor(Color.WHITE);
    vysledkyText.setTextColor(Color.WHITE);
    vysledkyText.setText("Výsledky měření:");
    spustTlacitko.setText("Spustit test");

    spustTlacitko.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        if(testBezi==false){
          casStartTestu = System.currentTimeMillis();
          testBezi=true;
        }
        else{
          vypisVarovani();
        }
      }
    });

    //přidání visuální prvků do layoutu
```

```
    relativeLayout.addView(mojeView);
    relativeLayout.addView(spustTlacitko);
    relativeLayout.addView(vysledkyText,parametryNahledu);

    setContentView(relativeLayout);  // nastaví náhled obrazovky na relativeLayout
  }

  //funkce pro výpis varování o probíhajícím testu
  public void vypisVarovani(){
    Toast toast = new Toast(this).makeText(this,"Test již probíhá !",
                                           Toast.LENGTH_SHORT);
    toast.show();
  }

  @Override
  protected void onPause() {
    super.onPause();
    mojeView.pause();
  }


  @Override
  protected void onResume() {
    super.onResume();
    mojeView.resume();
  }

  //Třída, která je použita jako náhled, na který je možno kreslit.
  public class MojeView extends SurfaceView implements Runnable{
    Thread t = null;
    SurfaceHolder holder;
    boolean isRunning = false;

    @SuppressWarnings("deprecation")
    public MojeView(Context context) {
      super(context);
      holder = getHolder();
    }

    public void run() {
      Paint paint = new Paint();
      paint.setColor(Color.RED);
      paint.setStrokeWidth(2);

      Canvas c;
      double uhel = 0;

      //Parametry testu
      int polomer=30; // poloměr v pixelech
      int krokuVKruhu = 240; //počet kroků na vykreslení jednoho kruhu
      double krokUhlu = (2*Math.PI)/(double)krokuVKruhu; // inkrement úhlu
      int maxOtocek = 5;

      while (isRunning == true){
        if (!holder.getSurface().isValid()){
          continue; //
        }
        c = holder.lockCanvas();
```

```
        if(testBezi==true){
          c.drawLine(200, 40, 200+FloatMath.sin((float)uhel)*polomer,
                         40+FloatMath.cos((float)uhel)*polomer, paint);

          uhel+=krokUhlu;
          if(uhel > 2*Math.PI){
            uhel=0;
            if(paint.getColor()==Color.RED)
              paint.setColor(Color.BLUE);
            else
              paint.setColor(Color.RED);

            pocetProvedenychOtocek++;

            if(pocetProvedenychOtocek==maxOtocek){ // kontroluje, zda test skončil
              casKonecTestu = System.currentTimeMillis();
              testBezi=false;
              pocetProvedenychOtocek=0;

              final long vysledek = casKonecTestu-casStartTestu;
              final int fps = Math.round((float)((double)1000 /
                    ((double)vysledek/(double)maxOtocek/(double)krokuVKruhu)));
                                        //vypočte počet snímků za sekundu

              runOnUiThread(new Runnable() { //změna zobrazení-nutně v UIvlákně
                public void run() {
                  vysledkyText.setText(vysledkyText.getText()+"\n"+vysledek+"ms,
                                       "+fps+"fps");
                }
              });
            }
          }
        }
        holder.unlockCanvasAndPost(c);
      }
    }
    // chrání vlákno před chybou při přerušení
    public void pause(){
      isRunning = false;
      while(true){
        try{
          t.join();
        }catch(InterruptedException e){
          e.printStackTrace();
        }
        break;
      }
      t = null;
    }
    // chrání vlákno před chybou při přerušení
    public void resume(){
      isRunning = true;
      t = new Thread(this);
      t.start();
    }
  }
}
```

Příloha C – Zdrojový kód aktivity – Zápis/čtení souboru SD karta – Java

```
package com.example.ctenizapissoubor;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;

import android.os.Bundle;
import android.os.Environment;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
  EditText editovatelnyText;
  Button tlacitkoZapsat;
  Button tlacitkoVypsat;
  TextView textovyPopisek;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    editovatelnyText = (EditText)findViewById(R.id.editText1);
    tlacitkoZapsat = (Button)findViewById(R.id.button1);
    tlacitkoVypsat = (Button)findViewById(R.id.button2);
    textovyPopisek = (TextView)findViewById(R.id.textView1);

    tlacitkoZapsat.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        String textKZapsani = new String();
        textKZapsani = editovatelnyText.getText().toString();

        if(!textKZapsani.equals("")){
          if(zapsatDoSouboru(textKZapsani)==false)
            vypisVarovani("Zápis do souboru na SD kartu se nezdařil!");
          else{
            vypisVarovani("Zápis do souboru na SD kartu byl proveden.");
            editovatelnyText.setText("");
          }
        }
      }
    });

    tlacitkoVypsat.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
```

```
      String textSouboru = precistSoubor();

      if(textSouboru!=null)
        textovyPopisek.setText(textSouboru);
      else
        vypisVarovani("Čtení souboru se nezdařilo");
    }
  });
}

//metoda pro výpis upozornění
public void vypisVarovani(String varovani){
  Toast toast = new Toast(this).makeText(this, varovani, Toast.LENGTH_SHORT);
  toast.show();
}

public boolean zapsatDoSouboru(String str){
  File soubor;
  BufferedWriter out;

  String cestaKSouboru=new String(Environment.getExternalStorageDirectory()+
                                  "/souborData.txt");
  soubor = new File(cestaKSouboru);

  if(!soubor.exists()){   //vytvoří soubor pokud neexistuje
    try {
      soubor.createNewFile();
    } catch (IOException e) {
      e.printStackTrace();
      return false;
    }
  }

  try { // připraví výstup k zápisu do souboru
    FileOutputStream souborOS =new FileOutputStream(soubor,true);
    OutputStreamWriter vystupWriter = new OutputStreamWriter(souborOS,
                                                    "windows-1250");
    out = new BufferedWriter(vystupWriter);
  } catch (UnsupportedEncodingException e) {
    e.printStackTrace();
    return false;
  } catch (FileNotFoundException e) {
    e.printStackTrace();
    return false;
  }

  try {  // pokusí se provést zápis do souboru
    out.write(str+"\n");
    out.flush();
    out.close();   // uzavře soubor
  } catch (IOException e) {
    e.printStackTrace();
    return false;
  }
  return true;
}

public String precistSoubor(){
  File soubor = null;
```

```
    BufferedReader in = null;
    String radek = new String();
    String obsahSouboru = new String();

    try {
      soubor = new File(Environment.getExternalStorageDirectory() +
                                                    "/souborData.txt" );
       FileInputStream souborIS = new FileInputStream(soubor);
       in = new BufferedReader(new InputStreamReader(souborIS, "windows-1250"));

    } catch (FileNotFoundException e2) {
      e2.printStackTrace();
      return null;
    } catch (UnsupportedEncodingException e) {
      e.printStackTrace();
      return null;
    }

    try {
      if (soubor.exists()){
        while((radek=in.readLine())!=null){
          obsahSouboru += radek+"\n";
        }
        in.close();
        soubor.delete();
      }
      else{
        in.close();
        return null;
      }
    } catch (IOException e) {
      e.printStackTrace();
      return null;
    }
    return obsahSouboru;
  }
}
```