Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Java aplikace - teorie a praxe

Robert Vokáč

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Robert Vokáč

Informatika

Název práce

Java aplikace – teorie a praxe

Název anglicky

Java applications – theory and practice

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku vývoje aplikací v jazyce Java. Hlavním cílem práce je charakterizovat základní aspekty, které jazyk Java programátorům nabízí.
Dílčí cíle bakalářské práce jsou:
- analyzovat obecné požadavky na samotný programovací jazyk,
- charakterizovat různé problémy, které s sebou vývoj aplikací přináší a ukázat možná řešení v praxi,
- navrhnout nejschůdnější možné řešení určitých problémů při návrhu a vývoji aplikací.
- zpracovat zkušenosti s vývojem vlastních aplikací.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Dále také na vlastních zkušenostech autora BP získaných při tvorbě aplikací.

Vlastní řešení je realizováno formou návrhů různých metod, kterými lze řešit různé situace a problémy.

Na základě syntézy teoretických poznatků, praktických zkušeností a výsledků vlastního řešení budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30 40 stran

Klíčová slova

Java aplikace, příkazy, vzory, modely, situace, možnosti, hlediska, plánování, návrh

---

Doporučené zdroje informací

ČADA, O. Objektové programování – naučte se pravidla objektového myšlení. Praha : Grada Publishing, a.s., 2009. 978-80-247-2745-5.

DARWIN, I., F. Java – Kuchařka programátora, Computer Press, 2006, 800 s. ISBN: 80-251-0944-5

ECKEL, B. Myslíme v jazyku Java : knihovna zkušeného programátora. 1. vyd. Praha : Grada Publishing, spol. s r.o., 2001. 472 s. ISBN 80-247-0027-1.

HEROUT, P. Java : grafické uživatelské prostředí a čeština. 1. vyd. České Budějovice : KOPP, 2001. 316 s. ISBN 80-7232-150-1.

HEROUT, P. Učebnice Jazyka Java. 2. vyd. České Budějovice : KOPP, 2006. 349 s. ISBN 80-7232-115-3.

KEOGH, J. Java bez předchozích znalostí. Brno : Computer Press, a.s., 2012. ISBN 978-80-251-0839-0.

KISZKA, B. 1001 tipů a triků pro jazyk Java. Brno : Computer Press, a.s., 2009. ISBN 978-80-251-2467-3

RISCHPATER, R. Beginning Java ME Platform, 2008, 600 s. ISBN:9781430210610

SCHILDT, H. Java 7 – Výukpvý kurz. Brno : Computer Press, a.s., 2012. ISBN 97880-251-3748-2.

SUBRAHMANYAM, A. Professional Java Server Programming J2EE, Apress, 1634 s. ISBN:9781861004659

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Čestmír Halbich, CSc.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 21. 10. 2016

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 10. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 12. 03. 2017

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Java aplikace - teorie a praxe" 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 14. 3. 2017 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Čestmíru Halbichovi, CSc. za vedení při zpracování bakalářské práce.

Java aplikace - teorie a praxe

Souhrn

Bakalářská práce se zaměřuje na problematiku vývoje aplikací v programovacím jazyce Java a sestává ze dvou hlavních částí, teoretické a praktické.

V teoretické části je popsána charakteristika Javy, možné problémy při vývoji aplikací a následně jejich možné řešení pomocí některých nástrojů. V charakteristice Javy je popsána její historie, současnost, předpověď možného budoucího vývoje, princip a syntaxe. Nástroji pro řešení problémů jsou testování, refaktorování, správa verzí, statická analýza kódu a dokumentace.

Praktická část je reprezentována logickou hrou Color Lines Sequel ve formě desktopové aplikace a víceúčelovou knihovnou Okay, které jsou zde popisovány. Color Lines Sequel i Okay jsou napsány v jazyce Java a ukazují praktické použití možností Javy uvedených v teoretické části.

Klíčová slova: Java, desktopová Java aplikace, JDBC, JavaFX, SQLite, relační databáze, objektově relační mapování, faktorování kódu, testování kódu, statická analýza kódu, logická desktopová hra, Java knihovna

Java applications - theory and practise

Summary

This bachelor thesis is focused on the issues of application development in Java programming language and consists of two main parts, theoretical and practical.

The theoretical part describes the characteristics of Java, possible problems with the development of applications and consequently their solutions by using some tools. The characteristics of Java describes its history, present and potential future, principles and syntax. Tools for solving problems are testing, refactoring, version control, static code analysis and documentation.

The practical part is represented by a puzzle game Color Lines Sequel in the form of desktop applications and by a multi-purpose library Okay, as described herein. Color Lines Sequel and Okay are written in Java and demonstrate the practical use of Java described in the theoretical part.

Keywords: Java, Java desktop application, JDBC, JavaFX, SQLite, relational database, object relational mapping, code refactoring, code testing, static code analysis, logic desktop game, Java library

Obsah

Seznam obrázků

1 Úvod

Programovací jazyk Java si již dlouhou dobu drží pozici nejoblíbenějšího a nejvíce používaného programovacího jazyka pro vývoj aplikací. Nejvíce je využíván pro tvorbu webových aplikací, umožnuje ale i tvořit desktopové a konzolové aplikace a aplikace pro mobilní telefony.

Mezi výhody Javy patří nezávislost na platformě a vysoká úroveň abstrakce kvůli objektovému přístupu. Mezi nevýhody patří nutnost instalace virtuálního stroje a režijní náklady během interpretace instrukcí virtuálního stroje na nativní instrukce procesoru. Záleží na konkrétní situaci, zda uživatelé budou ochotni instalovat virtuální stroj Javy a o jak moc bude rychlejší vývoj projektu, pokud bude použita Java místo například programovacího jazyka C++.

Cílem během vývoje aplikací by měla být snaha o co nejlepší architektonický návrh, o správně použitou abstrakci na jednotlivých vrstvách aplikace, o srozumitelný a do budoucna rozšiřitelný kód. Dostatečná abstrakce kódu by měla snížit složitost kódu, rozložit složitost a zodpovědnost, zrychlit vývoj nových funkčností a úpravu stávajících.

Téma jsem si vybral, protože programovací jazyk Java umožňuje vytvářet i desktopové aplikace. I přesto, že s pomocí standardní knihovny Javy, nástrojů pro ukládání trvalých dat a knihoven pro tvorbu grafického uživatelského rozhraní by mělo být možné vytvořit jakoukoliv desktopovou aplikaci, neexistuje tolik desktopových aplikací v Javě.

Během vývoje aplikací se objevují různé problémy, zároveň existuje mnoho nástrojů, které se snaží tyto problémy vyřešit. Pro programátory by mělo být důležité mít přehled o co nejvíce možnostech, které mají během programování, aby nebyli ochuzeni o různé nástroje jenom z důvodu nevědomosti jejich existence.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této bakalářské práce je charakterizovat možnosti, které programovací jazyk Java nabízí programátorům. Dílčími cíli jsou rozbor problémů, které se během vývoje aplikací vyskytují a následně navrhnout řešení těchto problémů. Dalším dílčím cílem je implementovat logickou hru ve formě desktopové aplikace a tím ukázat možnosti tvorby desktopových Java aplikací.

2.2 Metodika

Metodika této bakalářské práce je založena na studiu odborné literatury, odborných webových stránek a na vlastních zkušenostech autora. Pomocí analýzy možností, které programovací jazyk Java nabízí, jsou vybírány příslušné možnosti. Je využita abstrakce, kdy během implementace desktopové aplikace jsou problémy rozděleny do více úrovní a na každé úrovni je s problémem zacházeno jednodušeji skrytím některých detailů. Během vývoje aplikace jsou uskutečňovány experimenty implementace různých funkčností a následně jsou analyzovány, potom je rozhodnuto, zda je daná implementace správná, zcela špatná nebo zda úpravou bude správná. Dále je využíváno měření výkonnosti kódu a jeho následnému porovnání za jiných podmínek, testování, statická analýza kódu, verzování a systém pro správu úkolů. K vypracování této bakalářské práce byl použit software Netbeans IDE, Dia, SQLite Browser, Gitk, Microsoft Office a Libre Ofice. K formátování kódu byla využita webová aplikace https://tohtml.com/java/.

3 Teoretická východiska

3.1 Programovací jazyk Java

3.1.1 Historie

V roce 1991 vznikl programovací jazyk Oak. James Gosling byl jedním z jeho předních autorů. Po zjištění, že již existuje jiný programovací jazyk pojmenovaný Oak bylo rozhodnuto, že je nutné Oak přejmenovat. V roce 1995 se sešli přední autoři Oaku v jedné kavárně, aby vymysleli nový název. Vymysleli název Java, což je v americké angličtině slangový výraz pro kávu.
Důvodem ke zrození Javy byla neexistence multiplatformního programovacího jazyka. Jelikož vestavná zařízení byla ovládána softwary vytvořenými pomocí jazyků C a C+ a přenosnost těchto aplikací byla problematická, měla Java vyřešit tyto problémy. Prvotním důvodem vzniku Javy bylo vytvořit nový programovací jazyk pro přenosná zařízení, Java aplety byly následným faktorem, který umožnil růst oblíbenosti Javy. Použití apletů Javy jako aplikací nezávislých na platformě a běžících v internetovém prohlížeči umožnilo kromě přenositelnosti i možnost využití funkcí, které sám internetový prohlížeč neumožňoval.
Java ve své syntaxi vychází z jazyků C a C++, byla ovlivněna také Smalltalkem. Z důvodu rozporů mezi Sunem a Microsoftem přestala být Java automatickou součástí operačního systému Windows a vznikl ve firmě Microsoft jazyk C#. Jazyky C# a Java se dosud během svého vývoje vzájemně ovlivňují a inspirují.
Ačkoliv byla Java vyvinuta původně pro multiplatformní aplikace, je nyní převážně používána k tvorbě webových aplikací, kde je využíván aplikační model klient/server. Java zde běží na serverové části, která je vždy stejná a klientská část, kterou je webový prohlížeč, o Javě vůbec neví. [1] [2]

3.1.2 Současnost

Nejnovější verzí programovacího jazyka Java je verze 8. Jazyk se stále vyvíjí s každou další verzí,
Využití Javy jako programovacího jazyka pro aplikace operačního systému Android roste, naopak se utlumuje vývoj JavaME aplikací pro takzvané hloupé telefony, které jsou nahrazovány chytrými. Java je dále využívána zejména ke psaní webových aplikací jako podnikových informačních systémů, portálů bank a jiných.
Java aplety, které stojí za počátečním úspěchem Javy, přestávají být internetovými prohlížeči podporovány jako bezpečnostní hrozba a Oracle plánuje v Javě 9 plugin pro aplety odstranit. Jako náhrada apletů je doporučován nástroj Java Web Start, který umožňuje spouštět desktopové Java aplikace v okně prohlížeče. Dle žebříčku oblíbenosti programovacích jazyků Tiobe Programming Community Index si Java stále drží svoji vedoucí pozici, která je nyní 20%.[23]

3.1.3 Možný budoucí vývoj

Java jako programovací jazyk i platforma se stále vyvíjí. V červenci 2017 má vyjít Java SE verze 9 mající přinést například následující funkce:

- modularizace kódu

- jshell- přímé provádění příkazů Javy bez jejich zabalení do objektů a metod
- vylepšení aplikačního prostředí pro práci s procesy

V Javě SE verze 10 jsou plánovány nové knihovny pro práci s formátem json a pro práci s penězi a měnami.
Velký podíl podnikových webových Java aplikací dává velký prostor pro další existenci Javy, jelikož podniky raději upřednostňují ověřené technologie a nerady přepisují rozsáhlé informační systémy do jiného programovacího jazyka. Příkladem může být velké množství aplikací naprogramovaných v programovacím jazyce Cobol. [9] [12]

3.1.4 Vymezení

Java je multiplatformní. Zdrojový kód Javy není překládán do strojových instrukcí procesoru počítače ale do strojových instrukcí virtuálního stroje Javy. Běh Java aplikací nevyžaduje fyzicky existující procesor, pouze existenci virtuálního stroje Javy. Existuje více implementací virtuálního stroje stejné specifikace, například Java Virtual Machine, OpenJDK. Existují i některé implementace virtuálního stroje, které nejsou bezplatné a nabízejí výkonnostní nebo jiné výhody.
Programovací jazyk Java je multiparadigmatický, podporuje objektové, strukturované, imperativní, funkcionální, generické i paralelní paradigma. [1] [2]

3.1.4.1 Objektové paradigma

Pracuje s termíny objekt, třída, členové třídy, metody, abstrakce, zapouzdření, kompozice, dědičnost a polymorfismus.
Objekt je prvek modelované reality obsahující data i funkčnost.
Třída je typ objektu a obsahuje jeho definici. Objekty jsou takzvanými instancemi tříd.
Členové třídy jsou data vztahující se k objektu.
Metody jsou funkce vztahující se k objektu.
Abstrakce umožňuje rozdělit architekturu aplikace do více vrstev a oddělit podstatné od detailů.
Zapouzdření chrání stav objektů, stav můžou měnit pouze prvky s oprávněním.
Kompozice je skládání objektu z objektů dalších.
Dědičnost umožňuje objektu využívat schopností předka.
Polymorfismus definuje rozhraní pro komunikaci s objektem, následně se může pracovat s objekty majícími toto rozhraní stejné, i když mají různou implementaci. Polymorfismus dále umožňuje takzvané přetěžování metod se stejným názvem, které se ale liší pouze seznamem argumentů.

```
public interface MakingSound {

    public void makeASound();
}

class CatSoundMaker {

    private MakingSound cat;

    CatSoundMaker() {
        this.cat = new Cat();
        cat.makeASound();
    }
```

```
}

class Person implements MakingSound {

    @Override
    public void makeASound() {
        System.out.println("Good morning");
    }
}

class Car implements MakingSound {

    @Override
    public void makeASound() {
        System.out.println("Roar");
    }
}

class Cat implements MakingSound {

    @Override
    public void makeASound() {
        System.out.println("Meow");
    }

}
```

3.1.4.2 Strukturované paradigma

Tok kódu je pomocí řídících příkazů strukturován. Mezi řídící příkazy patří if, then, else, while, do while, switch, break, continue a return.

```
public class Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;
        if ((a + b) == c) {
            System.out.println("a");
            if (b < 10) {
                System.out.println("c");
            }
        } else if (a < c) {
            int i = 0;
            while (i < 3) {
                System.out.println("d");
            }
        }
    }
}
```

3.1.4.3 Imperativní paradigma

Kód je složen z příkazů oddělených znakem středník a není nijak větven.

```
public class Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;
        System.out.print(b);
        System.out.println(a);
        System.out.println(System.nanoTime());
    }
}
```

3.1.4.4 Funkcionální paradigma

Vyhodnocení kódu je chápáno jako vyhodnocení matematických funkcí.

```
import java.util.function.BinaryOperator;

public class Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        BinaryOperator<Integer> multiply = (a, b) -> a * b;
        Integer product = multiply.apply(1, 2);
    }
}
```

3.1.4.5 Generické paradigma

Algoritmus je oddělen od konkrétních typů, kód je zapsán obecně a konkrétní typ se definuje až při použití kódu dle dané situace.

```
class ValueHolder<T> {

    private T value;

    ValueHolder(T value) {
        this.value = value;
    }

    T getValue() {
        return this.value;
    }

    void setValue(T value) {
        this.value = value;
    }
```

3.1.4.6 Paralelní paradigma

Umožňuje provádět různý kód v jeden okamžik. V Javě se k tomuto využívají takzvaná vlákna. V jeden okamžik může být vykonáváno více částí kódu najednou, přináší to při správném použití nárůst výkonu. Řízení běhu vláken ale vyžaduje nutnou režii navíc, která výkon o něco snižuje.

```
public class Application {

    /**
     * @param args the command line arguments
     */
    public Application() {
        Thread thread1 = new Thread(this::write1, "One writer");
        Thread thread2 = new Thread(this::write2, "Two writer");
        thread1.start();
        thread2.start();
    }

    public static void main(String[] args) {
        Application application = new Application();
    }

    private void write1() {
        writeNTimes(1, 500);
    }

    private void write2() {
        writeNTimes(2, 500);
    }

    private void writeNTimes(int value, int count) {
        for (int i = 0; i < count; i++) {
            System.out.print(value);
        }
    }
}
```

3.1.5 Syntaxe

Syntaxe jazyka je soubor pravidel definujících kombinaci symbolů, pomocí kterých lze ověřit, zda daný jazyk dodržuje svoji syntaxi.
Syntaxe se skládá ze slov, frází a kontextu. Slova určují kombinace znaků. Fráze jsou určovány kombinací slov. Kontext určuje celkovou funkčnost pomocí vztahů.
Prvky syntaxe Javy jsou proměnná, operátor, výraz, příkaz, klíčové slovo a data. [1] [2]

3.1.5.1 Slova v Javě

- literály- primitivní datové typy, například celá čísla nebo znaky
- proměnné pojmenovávající data
- operátory provádějící operace s daty
- klíčová slova, což jsou rezervovaná slova, která nelze použít pro názvy proměnných

3.1.5.2 Fráze v Javě

- výraz, což je konstrukce složená z proměnných a operátorů
- příkaz, což je instrukce ukončená středníkem a obsahující volání metod, řízení běhu kódu, přiřazení výrazu proměnné nebo vytváření nových objektů.

3.1.5.3 Typový systém v Javě

Java je silně typovaný jazyk. Každá proměnná musí mít definován datový typ dat, která se budou do proměnné ukládat.
Datovým typem může být primitivní typ, objekt, pole nebo rozhraní

3.1.5.4 Primitivní typy

Mají pevnou velikost, obvykle se jich vyskytuje mnoho, zabírají málo místa, pracuje se s nimi rychleji než s objekty a jsou ukládány do zásobníku.
Primitivní typy v Javě jsou:

- Pro celá čísla: int, long, byte, short
- Pro reálná čísla: float, double
- Pro znak: char
- Pro pravdivost: boolean

Java pro výpočty s celými čísly používá pouze datové typy int nebo long. Pokud je datový typ nějaké proměnné definován jako byte nebo short, Java převede tuto proměnnou na datový typ int předtím, než s ní začne pracovat. Tímto se ztrácí případná výhoda rychlejšího výpočtu, a vyplývá pouze výhoda využití menší velikosti paměti.

3.1.5.5 Třída

Třída je typ objektu a obsahuje definici tohoto typu. Třída má předka, jehož schopnosti využívá. Pokud třída nedefinuje svého předka, je jím automaticky třída Object, která je nejvzdálenějším předkem všech tříd. Třídy jsou definovány v textových souborech s příponou java. Jeden soubor s příponou java a se zdrojovým kódem v Javě může obsahovat maximálně jednu třídu s veřejnou přístupností a dále nekonečně mnoho tříd jiné přístupnosti. Pokud soubor java obsahuje veřejnou třídu, musí být název tohoto souboru shodný s názvem třídy. Soubor s příponou java obsahuje nejprve příkaz package definující balíček, v kterém se třída nachází, dále příkazy import umožňující ve třídě používat třídy z jiných balíčků. Následuje definice jedné nebo více tříd:

```
package utility;

import utility.math.Calculator;

public class Writer {

private final int number;

public Writer(int number)
{
this.number=number;
}
```

```
    public static void main(String[] args) {
        System.out.println("Test");
    }
}
```

3.1.5.6 Balíček

Třídy jsou organizovány do takzvaných balíčků. Balíček může obsahovat třídy, ale i další balíčky. Balíček je v souborové struktuře reprezentován složkou, jejíž název a umístění se musí shodovat s balíčkem.

3.1.5.7 Objekt

Objekty jsou typy dat, která mají data i funkčnost a jsou definovány v třídách. Objekty se získávají vytvořením nové instance třídy.

```
Animal animal = new Animal();
```

3.1.5.8 Pole

Pole je datová struktura, která má konečný počet prvků stejného typu. K jednotlivým prvkům se přistupuje pomocí indexu, kterým je celé číslo začínající od nuly. Pokud je například definováno pole o 10 prvcích, přistupuje se k prvnímu prvku pomocí indexu nula. Stejná velikost prvků pole umožňuje rychlejší přístup k těmto prvkům oproti kolekcí s prvky různé velikosti.

```
int [] anArray = new int [10];
anArray[0] = 8;
```

3.1.5.9 Rozhraní

Rozhraní definuje názvy metod spolu s jejich signaturami. Pokud nějaká třída implementuje rozhraní, znamená to, že implementuje všechny metody tohoto rozhraní. Do proměnných, které mají jako datový typ nějaké rozhraní, lze ukládat všechny objekty, jejichž třídy toto rozhraní implementují.

```
public interface AInterface
{
public void a();
}
```

3.1.5.10 Přístupnost

Třídy, třídní proměnné a metody mají vždy definovanou svoji přístupnost pro ostatní prvky.
Přístupnost je definována pomocí klíčových slov public, protected, private nebo prázdného řetězce. Přístupnost private, znamená přístupnost všem, protected pouze předkům a třídám ze stejného balíčku, private pouze členům třídy a prázdný řetězec pouze třídám ze stejného balíčku. Přístupnost se nastaví přidáním patřičného klíčového slova v definici třídy, proměnné nebo metody. Přístupnost package reprezentovaná prázdným řetězcem se nezapisuje a je implicitní. Není možné zajistit přístupnost pouze pro předky, jelikož v přístupnosti package jsou zahrnuty i dědící třídy.

3.1.5.11 Statika

U datových členů tříd a metod existuje vlastnost statika. Pokud je datový člen třídy nebo metoda statický, znamená to, že data přísluší k třídě nebo, že metoda se volá bez vytvoření instance. Statika se zajistí přidáním klíčového slova static v definici. Statické datové členy existují bez nutnosti vytvořit nějakou instanci třídy. Statické metody se nevolají na instancích ale na třídách. Typickým příkladem vhodnosti použití statiky je třída Math ze standardní knihovny Javy. Bylo by nepraktické a neefektivní vytvářet při každé operaci novou instanci třídy Math.

```
Double number = -10.5;
Systém.out.println(Math.abs(number));
```

3.1.6 Použití Javy

Knihovna Javy se dělí dle svého rozsahu na několik edic. Edice JavaME, JavaSE, JavaEE. [1][2]

3.1.6.1 Desktopové aplikace

Java je pro vývoj desktopových aplikací využívána málo, jelikož kladou na uživatele požadavek instalace virtuálního stroje. Příkladem desktopové aplikace v Javě je vývojové integrované prostředí Netbeans. Desktopové aplikace jsou postaveny na edici JavaSE.

3.1.6.2 Webové aplikace

Webové aplikace využívají aplikační model klient-server. Na serverové části běží Java, která dynamicky generuje webové stránky. Na klientské části Java neběží, klientská část, kterou je internetový prohlížeč, zpracovává dynamicky vygenerované webové stránky na serveru pomocí například značkovacího jazyka HTML a skriptovacího jazyka JavaScript. Webové Java aplikace jsou postaveny na edici JavaEE.

3.1.6.3 Aplikace na operačním systému Android

Aplikace vytvořené v programovacím jazyku Java pro operační Android nemohou využívat celou standardní knihovnu Javy, ale pouze její podmnožinu. Zdrojový kód v Javě aplikací pro Android je nejprve zkompilován do bytekódu, bytekód je následně zkompilován do instrukcí virtuálního stroje Androidu Dalviku nebo novějšího Android Runtime. Implementace Javy na Androidu není nijak podřízená firmě Oracle, která je vlastníkem Javy. Na operačním systému Android například nelze využívat rozhraní JDBC pro databáze, mobilní zařízení svým charakterem ani neodpovídá databázovému serveru. Na Androidu je však možné využít pomocí vlastního knihovny aplikačního rozhraní Androidu odlehčenou databázi SQLite a ukládat tak data pomocí relačního modelu.

3.1.6.4 Aplikaci pro zařízení s omezenou pamětí

Pro zařízení s vyloženě znatelně omezenou velikostí paměti je také používána JaveME, která je podmnožinou standardní knihovny Javy. JavaME je využívána na takzvaných hloupých telefonech, ale i v tiskárnách, televizích a settopboxech. S růstem výkonu

výpočetních zařízení a podílu operačního systému Android je JavaME nahrazována Javou operačního systému Android.

3.2 Způsoby ukládání dat trvalého charakteru

Skoro všechny vyvíjené aplikace bez ohledu na použitý programovací jazyk mají za cíl nějaká data převzít, zpracovat, ukázat a uložit. Ukládání dat, která nemají trvalý charakter, lze zařadit jako zpracování. Data, u kterých je požadována jejich existence i po vypnutí aplikace a jejím opětovném spuštění, mají trvalý charakter. Data lze trvale ukládat různými způsoby, je možné je uložit do souboru, do databáze, na vzdálený server pomocí sítě nebo využít nějakou knihovnu, která bude sama zajišťovat způsob uložení dat.

3.2.1 Do souboru

3.2.1.1 Jako prostý text

Prostý text ukládaný a načítaný ze souborů musí mít nějakou strukturu, aby se s ním mohlo pracovat.
Příklady formátů textových souborů s určitou strukturou jsou značkovací jazyky (xml, html, svg), json nebo csv. Nejjednodušší strukturou je řádková struktura, příkladem je zaznamenávání chodu aplikace do logovacích souborů. Textový soubor se skládá ze znaků. Znaky jsou kódovány pomocí nějaké znakové sady, kde je každému znaku přiděleno nějaké číslo, znak je ukládán pomocí tohoto čísla a při následném čtení jsou čísla interpretovány na znaky. Mezi nejznámějšími znakovými sadami jsou Ascii pro jazyky využívajícími pouze latinku a Unicode, které je univerzální s ohledem na množství možných znaků, které lze zakódovat. [1] [2]

3.2.1.2 Příklad zápisu a čteni textového souboru v Javě

```
package textfilesreadingandwriting;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;

/**
 *
 * @author Robert Vokáč e-mail: robertvokac1@gmail.com
 */
public class TextFilesReadingAndWriting {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
// Writing text file
        System.out.println("Writing file: ");
        try (BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter("file.txt"))) {
            bufferedWriter.write("Row 1\nRow 2\nRow 3\nRow 4");
            bufferedWriter.flush();
            System.out.println("Writing was successful");
```

```
        } catch (Exception exception) {
            System.err.println("Writing was not successful");
        }
// Reading text file
        System.out.println();
        System.out.println("Reading file: \n");
        try (BufferedReader bufferedReader = new BufferedReader(new
FileReader("file.txt"))) {
            String string;
            while ((string = bufferedReader.readLine()) != null) {
                System.out.println(string);
            }
        } catch (Exception exception) {
            System.err.println("An error occured while reading");
        }
    }
}
```

3.2.1.3 Formát JSON

Formát JSON je textový formát k výměně dat. Je srozumitelný pro člověka a snadno zpracovatelný počítači Zkratka JSON vychází z anglického JavaScript Object Notation, tedy JavaScriptová objektová notace. Formát JSON je orientovaný spíše pro přenos dat v objektové podobě na rozdíl od značkovacího jazyka XML, které je orientováno spíše dokumentově. Ačkoliv formát JSON začal být prvotně využíván v programovacím jazyce JavaScript, je JSON jazykově nezávislý a lze s ním pracovat v mnoha programovacích jazycích.
V Javě je možné pracovat s formátem JSON pomocí různých knihoven třetích stran, například MinimalJson, OrgJson, Gson, Jackson nebo JsonSimple. Knihovna pro práci s formátem JSON ve standardní knihovně Javy v současné době neexistuje, ale pracuje se již na ní a měla by se objevit v Javě verze 10 nebo pozdější.
Formát JSON je postaven na dvou strukturách, objektu a poli. Objekty a pole mají hodnoty. Hodnotami jsou objekty, pole, řetězce, čísla, hodnoty pravdivost true a false nebo hodnota null.
Objekt je neuspořádaná kolekce a je ohraničen složenými závorkami { a }. Uvnitř složených závorek se umisťují jednotlivé hodnoty objektu adresované klíči a oddělené čárkami.
Pole je uspořádaná kolekce a je ohraničena hranatými závorkami [a]. Uvnitř hranatých závorek se umisťují hodnoty pole oddělené čárkami.
Existuje několik norem pro formát JSON, například ECMA-404 The JSON Data Interchange Standard. Tyto normy obsahují pokročilé a detailní možnosti formátu JSON.

Příklad textového souboru formátu JSON:

```
{
    "cpu":"Intel Corei7",
    "cores":4,
    "desktop":false,
    "ram":{
        "capacity":8,
        "units":"GB"
    },
```

```
    "monitors":[
        {
            "brand":"Dell",
            "width px":1920,
            "height px":1080
        },
        {
            "brand":"Samsung",
            "width px":1024,
            "height px":768
        }
    ]
}
```

JSON objekty lze validovat, pokud je k nim přiloženo takzvané schéma. Schématem k nějakému JSON objektu je většinou JSON objekt, který má speciální strukturu popisující přípustnou podobu validovaného JSON objektu. Pomocí validace JSON objektu lze ověřit správnou podobu objektu ještě před jeho zpracováním. [12]

3.2.1.4 V binární formě

Se soubory v binárním režimu se nepracuje pomocí znaků ale přímo na úrovni nul a jedniček a umožňuje například implementovat vlastní formát souboru a využít tím maximálně efektivně a dle potřeby kapacitu úložného zařízení. Binární soubory ale na rozdíl od textových nejsou srozumitelné člověku. [2]

3.2.2 Do relační databáze

Databáze využívající relační model dat jsou stále velmi využívané, ačkoliv existují i databáze využívající jiný model dat, například dokumentově orientované nebo objektově orientované. Relační databáze oproti dvěma zbývajícím jmenovaným svazují k použití relačního modelu dat používajícího termíny jako tabulka, záznam, primární klíč, cizí klíč a dalším. Relační model umožňuje jednoduchou a striktní strukturu dat, ale naopak omezuje ve volnosti.

3.2.2.1 Princip práce s relačními databázemi v Javě.

Standardní knihovna Javy umožňuje aplikacím v Javě využívat relační databáze různých typů pomocí takzvaného JDBC rozhraní. K tomu, aby bylo možné v Javě využívat nějakou konkrétní relační databázi, pouze stačí existence takzvaného JDBC ovladače. JDBC ovladač je v podstatě knihovna, která implementuje rozhraní JDBC pro přístup ke konkrétnímu typu databáze, jelikož mezi přístupem k databázím různých typů existují rozdíly. Existují také JDBC ovladače pro přístup k datům, které nejsou uložené ve formátu relačního modelu, například sešit Microsoft Excel nebo formát csv obsahující tabulku v textové podobě, kde sloupce jsou odděleny pomocí nějakého znaku, většinou čárky, nebo dokumentově orientovaná databáze MongoDB.

3.2.2.2 Proces práce s databází v Javě pomocí rozhraní JDBC

Během práce s databází se musí dodržet několik kroků v přesném pořadí.
Proces práce s databází pomocí rozhraní JDBC zahrnuje zavedení ovladače JDBC, připojení k systému řízení báze dat, tedy konkrétní databázi, vytvoření a spuštění dotazu SQL, zpracování odpovědi, a odpojení ze systému. Vytvoření dotazu a získání odpovědi je možno opakovat více krát. [2]

3.2.2.3 Příklady tříd rozhraní JDBC

Třída **Connection** reprezentuje připojení k databázi, toto připojení lze mimo jiné uzavřít nebo vytvořit pomocí něj instanci třídy Statement.
Třída **Statement** je určena ke spouštění příkazů jazyka SQL a je znovu použitelná. Po spuštění SQL příkazu ji lze použít znovu, jelikož jednotlivé SQL příkazy reprezentované instancemi řetězců se instancím třídy Statement předávají jako parametry metod a ne jako parametry konstruktoru.
Třída **PreparedStatement** slouží k vytvoření parametrizovaných SQL příkazů. Tento příkaz je databázovým stroje jednou zkompilován a při každém použití se instanci třídy PreparedStatement předají konkrétní data, což umožňuje větší znovu použitelnost a zvýšení výkonu.
ResultSet reprezentuje výsledek nějakého SQL příkazu, kdy se vrací sada řádků majících nějaké sloupce.
Třída **DatabaseMetadata** reprezentuje přístup k různým metadatům o připojení k databází, kterými například jsou jména tabulek, primárních klíčů nebo schémata tabulek.
Třída **ResultSetMetaData** se využívá k získání různých informací nějaké instance ResultSet, kterými například jsou jména sloupců nebo počet sloupců.

3.2.2.4 Databáze SQLite

SQLite je relační databáze, která nepoužívá klasickou architekturu klient/server, jelikož SQLite není databázový server ale knihovna. Tato knihovna ukládá data databázi do jednoho souboru. Jelikož SQLite není server ale knihovna, znamená to výhody nulové konfigurace. SQLite běží jako proces pracující nad souborem.
SQLite databáze byla vytvořena D. Richardem Hippem v roce 2000, když D. Richard Hipp pracoval pro americké námořnictvo a snažil se vyvinout způsob práce nad relačním modelem dat bez nutnosti databázového serveru a jeho konfigurace.
SQLite je naprogramovaná v jazyce C. V některých případech může být použití SQLite databáze rychlejší než obyčejný přístup k souborům. V některých oblastech je SQLite rychlejší, než jiné databáze.
Existuje také JavaScriptová knihovna sql.js, která vznikla zkompilování kódu SQLite v jazyce C pomocí nástroje emcripten do jazyka Javascript. Knihovna sql umožňuje použít SQLite databázi ve webových aplikacích na straně klienta. [16]
SQLite implementuje většinu standardu SQL verze 92 s výjimkou například práv a některých druhů příkazu JOIN pro spojování tabulek a některých dalších pokročilých příkazů. Jelikož SQLite není server ale proces pracující nad souborem, byla by implementace práv nelogická, navíc samy soubory na disku mají nastavena svá přístupová práva.

SQLite databáze umožňuje ve schématu definovat datové typy sloupců, ale při operacích s tabulkami typy dat nekontroluje. Uvádí se, že toto není chyba, ale vlastnost. Neexistence kontroly typu vkládaných dat lze vyřešit přidáním omezení CHECK ve schématu tabulky u příslušného sloupce.

```
CHECK(typeof(name = ' text'))
```

3.2.2.5 Příklad práce s databází v Javě

Následující příklad demonstruje možnost připojit se v Javě k odlehčené relační databázi SQLite. Serverové typy databází navíc využívají k připojení údaje, kterými jsou uživatelské jméno a heslo.

```
package jdbcexample;

import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JdbcExample {

    private static Connection connection;
    private static Statement statement;
    private static String jdbcUrl;

    public static void main(String[] args) {
        jdbcUrl = "jdbc:sqlite:data.sqlite";
        ResultSet resultSet = null;
        try {
            //Loading JDBC driver
            Class.forName("org.sqlite.JDBC");
            //Creating database connection
            connection = DriverManager.getConnection(jdbcUrl);
            //Creating statement
            statement = connection.createStatement();
            //Executing statements without answer
            statement.execute("CREATE TABLE customer ("
                    + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                    + "first_name TEXT,"
                    + "last_name TEXT)");
            statement.execute("INSERT INTO customer VALUES
(null,'John','Patient')");
            statement.execute("INSERT INTO customer VALUES
(null,'Jack','Green')");
            statement.execute("INSERT INTO customer VALUES
(null,'Anne','Smith')");
            statement.execute("INSERT INTO customer VALUES
(null,'George','Smart')");
            //Executing statement with answer
            resultSet = statement.executeQuery("SELECT * FROM customer");
            //Getting result
            while (resultSet.next()) {
                System.out.println(
                        resultSet.getInt("id") + ", "
                        + resultSet.getString("first_name") + ", "
                        + resultSet.getString("last_name")
                );
            }
```

```
            //Droping table customer
            statement.execute("DROP TABLE customer");
        } catch (Exception e) {

Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE, null, e);
        } finally {
            //Closing connection
            try {
                if (resultSet != null) {
                    resultSet.close();
                }
                if (!connection.isClosed()) {
                    connection.close();
                }
            } catch (SQLException ex) {

Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }
    }
}
```

3.3 Tvorba grafického uživatelského rozhraní desktopových aplikací v Javě

K tvorbě grafického uživatelského prostředí desktopových aplikací v Javě mohou být využity knihovny standardní edice Javy, kterými jsou AWT, Swing a JavaFX nebo knihovny třetích stran, kterými jsou například Qt Jambi či knihovny postavené nad knihovnou OpenGL jako LWJGL a JOGL.

3.3.1 AWT

Abstract Window Toolkit je platformově závislá knihovna, tímto porušuje jeden z principů Javy. Když se zjistilo, že AWT obsahuje chyby v návrhu, byl ukončen vývoj. Uživatelské prostředí se vytváří pomocí předpřipravených komponent a složitější grafika pomocí bitmapové grafiky. Vzhled aplikací využívajících AWT se na jednotlivých platformách operačních systémů liší, jelikož AWT využívá nativní komponenty operačního systému místo vlastní implementace. [2]

3.3.2 Swing

Swing částečně využívá knihovnu AWT, oproti které nabízí bohatší možnosti. Swing například umožňuje vyměnit za běhu aplikace její vzhled. [2]

3.3.3 JavaFX

JavaFX je nejnovější oficiální knihovna pro tvorbu uživatelského rozhraní desktopových aplikací v Javě. Částečně se JavaFX inspirovala starším Swingem, který má nahradit. JavaFX vznikla nejprve jako projekt F3, F3 byl v roce 2007 přejmenován na JavaFX. JavaFX nejprve používalo k tvorbě grafiky svůj vlastní skriptovací jazyk, ve verzi JavaFX byl však tento skriptovací jazyk zcela odstraněn a nahrazen Javou. Oproti starší knihovně

Swing nabízí vyšší abstrakci a bohatší možnosti. JavaFX 8 je nejnovější verzí, ale v současné době se již pracuje v souvislosti s Javou 9 na JavěFX 9. [3]

3.3.3.1 Podpora JavyFX

JavaFX platforma je oficiálně podporována na operačních systémech Windows, Macintosh a Linux platformy x86. V minulosti existovala i podpora pro zařízení s operačním systémem Linux s procesorem ARM, tato podpora byla v lednu 2015 stažena. I přesto je například na minipočítači Raspberry Pi s operačním systém Raspbian s Linuxovým jádrem a s procesorem Arm prostřednictvím neoficiálních postupů možné spouštět Java aplikace používající platformu JavaFX. Na operačním systému Android je možné používat k tvorbě grafického uživatelského rozhraní platformu JavaFX, pokud je využita knihovna Gluon Mobile. Příkladem je Android aplikace Ensemble8, která demonstruje možnosti JavyFX a je dostupná v obchodu aplikací Googla Play pro Android. [3] [14] [15]

3.3.3.2 Princip JavyFX

Jelikož je JavaFX knihovna Javy, přistupuje se k tvorbě grafického uživatelského rozhraní pomocí objektového paradigmatu Javy. K tvorbě uživatelského prostředí lze ale i využít značkovací jazyk fxml nebo aplikaci Scene Builder.
V JavěFX se vyskytují tři základní termíny, kterými jsou jeviště, scéna a uzel
Pojmenování dvou základních termínů vyskytujících se během tvorby JavaFX aplikace vychází z terminologie divadla. K reprezentaci okna aplikace se používá třída Stage, čili jeviště. K reprezentaci scény jeviště, tedy toho, co se zobrazí v okně, se používá třída Scene.
Uzel je grafický prvek. Grafické prvky mají rodiče a mohou mít i potomky dle typu.
Scéna obsahuje stromový graf grafických prvků. [3]

3.3.3.3 Uzle

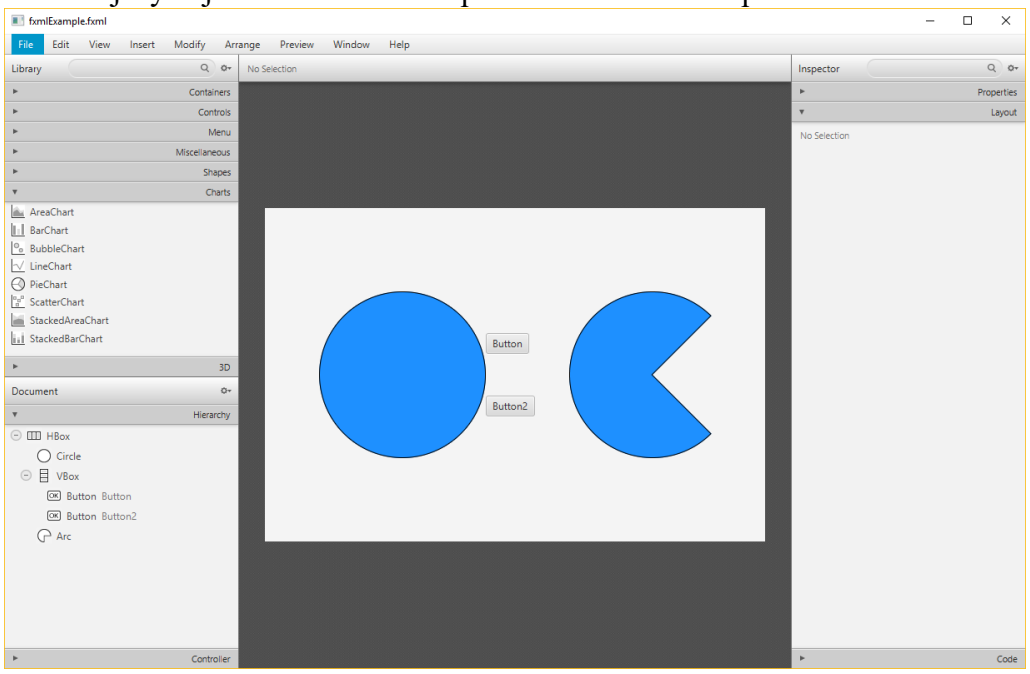
Uzle reprezentují většinou entity, které mohou být nějak vidět, například uzle rozložení, tvary, prvky ovládání.
Uzle reprezentující transformace a animace ale vidět nejsou jako jednotlivá entita, ale jako nějaký efekt aplikovaný na viditelný uzel či uzle. [3]

3.3.3.4 Další možnosti JavyFX

Dalšími možnostmi JavyFX jsou stylování uzlů pomocí CSS stylů, obsluha událostí, prohlížení webových stránek, práce s trojrozměrným prostředím, efekty, transformace-posun, rotace a škálování, animace, grafy, práce s bitmapovou grafikou, paralelní programování v JavěFX, přehrávání zvuků a videí, značkovací jazyk FXML k definování prostředí značkovacím jazykem místo Javy, rozhraní pro práci s tiskem a tiskárnou. Tyto funkce dávají prostor k tvorbě aplikací s rozmanitými možnostmi a vzhledem. [3]

3.3.3.5 Příklad tvorby grafického uživatelského rozhraní pomocí JavyFX a aplikace Scene Builder

Aplikace Scene Builder je určena k tvorbě fxml souborů pomocí vizuálních operací táhni a pust. Výsledky jsou ukládány do fxml souborů. FXML je značkovací jazyk, využití značkovacího jazyka je kvůli stromové reprezentaci vizuálních prvků velmi vhodné.



Obrázek 1: Příklad použití aplikace SceneBuilder

Zdroj: aplikace SceneBuilder obsahující vlastní práci autora

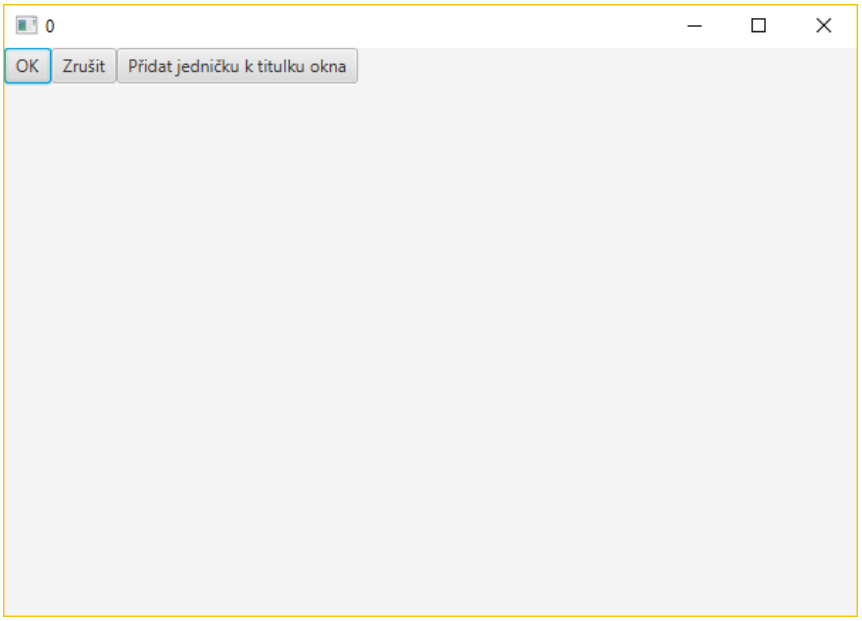
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.shape.Arc?>
<?import javafx.scene.shape.Circle?>


<HBox alignment="CENTER" maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0"
prefWidth="600.0" xmlns="http://javafx.com/javafx/8.0.102"
xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Circle fill="DODGERBLUE" radius="100.0" stroke="BLACK"
strokeType="INSIDE" />
      <VBox alignment="CENTER_LEFT" prefHeight="200.0" prefWidth="100.0"
spacing="50.0">
         <children>
            <Button mnemonicParsing="false" text="Button" />
            <Button mnemonicParsing="false" text="Button2"
textAlignment="RIGHT" />
         </children>
         <opaqueInsets>
            <Insets />
```

```
            </opaqueInsets>
        </VBox>
        <Arc fill="DODGERBLUE" length="270.0" radiusX="100.0"
radiusY="100.0" startAngle="45.0" stroke="BLACK" strokeType="INSIDE"
type="ROUND" />
    </children>
</HBox>
```

3.3.3.6 Příklad tvorby grafického uživatelského rozhraní pomocí JavyFX a Javy

Následující kód demonstruje jednoduchým způsobem vytvoření velmi primitivní Java aplikace využívající platformu JavaFX.

```
package javafxexample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class JavaFXExample extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("0");
        Button okButton = new Button();
        okButton.setText("OK");
        Button cancelButton = new Button("Zrušit");
        Button addOneButton = new Button("Přidat jedničku k titulku
okna");
        HBox root = new HBox();
        root.getChildren().addAll(okButton, cancelButton, addOneButton);

        okButton.setOnAction((ActionEvent event) -> {
            primaryStage.close();
        });

        cancelButton.setOnAction((ActionEvent event) -> {
            primaryStage.setTitle("0");
        });
        addOneButton.setOnAction((ActionEvent event) -> {
            String currentTitle = primaryStage.getTitle();
            int number = Integer.parseInt(currentTitle) + 1;
            primaryStage.setTitle(String.valueOf(number));
        });
        Scene scene = new Scene(root, 600, 400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



Obrázek 2: Ukázková aplikace využívající JavuFX
Zdroj: vlastní práce autora

3.4 Návrhové vzory

Návrhové vzory jsou doporučovaná řešení opakujících se problémů. Pokud programátor řeší nějaký problém, použití návrhového vzoru jej může vyřešit. Použitím návrhového vzoru lze využít osvědčené postupy řešení a vyhnout se případným zásadním chybám. Navíc si programátor může správně formulovat své programátorské myšlení pro řešení budoucích problémů. [4]

3.4.1 Rozdělení návrhových vzorů

3.4.1.1 Tvořivé

Tvořivé návrhové vzory slouží k vytváření instancí objektů jinými způsoby, než je prosté vytvoření instance přímo pomocí konstruktoru.
Návrhový vzor Stavitel odděluje konstrukci objektu od jeho reprezentace. Pokud existuje například třída House, implementace vzoru stavitel znamená vytvoření další třídy HouseBuilder. Třída HouseBuilder má sadu metod ke konstrukci třídy House, tyto metody vracejí zpět třídu HouseBuilder. Až voláním metody build je vracena instance třídy House.

3.4.1.2 Strukturální

Strukturální návrhové vzory skládat objekty k sobě.
Adaptér je návrhový vzor, který se použije, pokus je požadováno pracovat s nějakým objektem, ale daný objekt má jiné rozhraní metod, než je potřeba. Vytvoří se rozhraní umožňující komunikovat s daným objektem, toto rozhraní funguje v roli prostředníka.

3.4.1.3 Behaviorální

Behaviorální vzory slouží ke komunikaci mezi objekty.
Neměnný objekt je návrhový vzor, kdy je třída definována tak, že její stav lze určit pouze při inicializaci v konstruktoru. Příkladem neměnného objektu v Javě je třída String reprezentující textový řetězec. Řetězce v Javě nelze po vytvoření změnit, pouze lze vytvořit novou instanci třídy řetězec s příslušnými změnami, což ale přináší určité výkonnostní náklady.

3.5 Nástroje usnadňující vývoj aplikací v Javě

3.5.1 Dokumentace

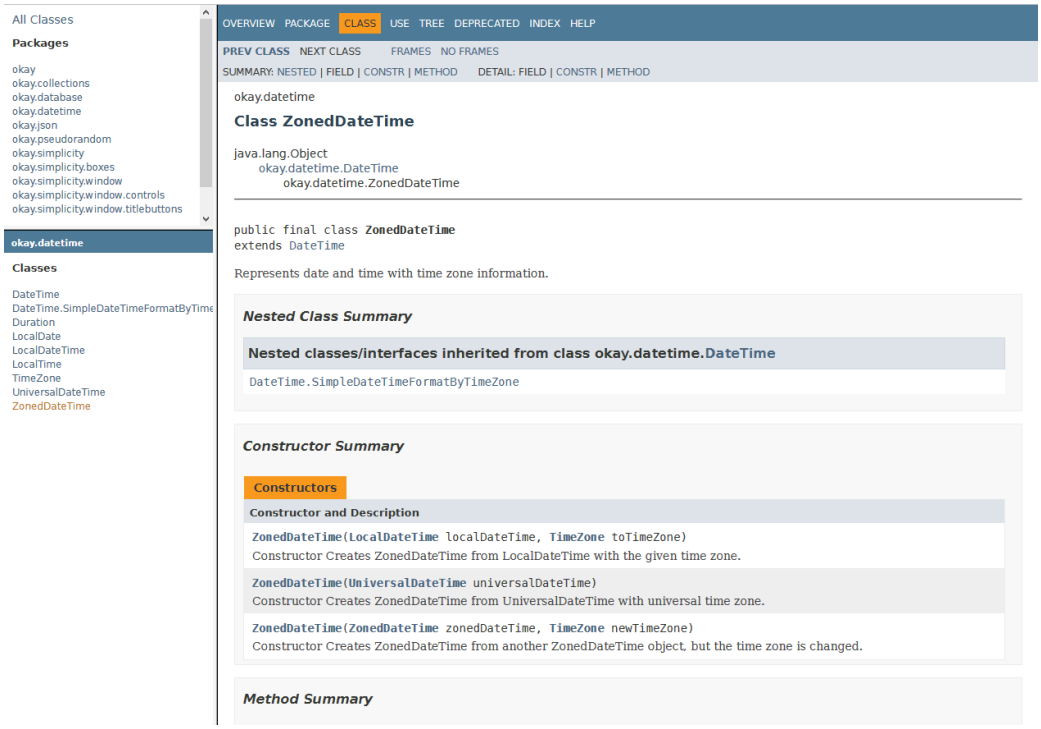
Během různých fází vývoje softwaru vznikají různé typy dokumentů. Mezi tyto dokumenty lze zařadit požadavky zákazníka, analýzu těchto požadavků a dokumentaci vztahující se k samotné implementaci. Implementační dokumenty můžou zahrnovat implementační příručku, různé diagramy a schémata procesů, návrhy uživatelského rozhraní. Během implementace mohou být využity i takzvané UML diagramy, které v grafické podobě zobrazují návrh nebo implementaci na různých úrovních abstrakce. Dalším dokumentem je uživatelská příručka, která popisuje software z uživatelského hlediska. Na úrovni kódu slouží k dokumentaci také komentáře.

3.5.1.1 Komentáře v Javě

V Javě existují tři druhy komentářů, jednořádkové začínající //, více řádkové začínající /* a zakončené */ a dokumentační komentáře začínající /* a zakončené **/. Dokumentační komentář je speciálním druhem komentáře metod nebo tříd a obsahuje jazyk JavaDoc. JavaDoc komentáře jsou určeny k vygenerování dokumentace ze zdrojového kódu v podobě webových stránek. [1] [2]

3.5.1.2 Příklad JavaDoc komentáře

```
   /**
     * Constructor
     *
     * Creates new LocalDateTime with these parameters.
     *
     * @param year
     * @param month
     * @param day
     * @param hour24Format
     * @param minute
     * @param second
     * @param millisecond
     */
    public LocalDateTime(int year, int month, int day, int hour24Format,
int minute, int second, int millisecond) {
        super(year, month, day, hour24Format, minute, second,
millisecond);
    }
```



Obrázek 3: JavaDoc dokumentace knihovny Okay

Zdroj: JavaDoc dokumentace vygenerovaná z vlastní práce autora

3.5.2 Konvence v psaní kódu

Konvence jsou způsoby psaní kódu, které nemají vliv na funkčnost aplikace a ani na implementaci, ale na podobu kódu. Některé doporučené konvence pocházejí od tvůrců Javy, jiné od autorů snažících se předat své zkušenosti a další konvence lze dohodnout pro práci v týmu během vývoje nějaké aplikace. Smyslem dodržování konvence je, aby kód napříč celou aplikací vypadal stejně. [5]

3.5.2.1 Konvence autorů Javy

Mezi konvence od autorů Javy patří například způsob pojmenovávání tříd, metod a balíčků a konstant. Jména tříd, metod a balíčků by neměla používat pro oddělování slov znak podtržítko „_“ ani žádný jiný. Pro oddělení slov u tříd a metod by mělo být každé slovo uvozeno velkým písmenem. Jméno třídy by mělo začínat velkým písmenem, jméno metody by mělo začínat malým písmenem. Balíčky by měly v názvu obsahovat pouze malá písmena. [22]

Konstanty jsou proměnné modifikované klíčovým slovem final, nemohou mít po přiřazení změněnou hodnotu. Toto chování se zdůrazňuje konvencí pro konstanty v Javě. Slova v názvu konstanty se oddělují znakem podtržítko „_“ a obsahují písmena všechna velká.

Konvencí je i pořadí deklarace členských proměnných a metod ve třídě. V deklaraci třídy by měly být nejdříve deklarovány statické proměnné, potom proměnné instanční. Dále by toto pořadí mělo být ovlivněno i přístupností. Nejdříve by se měly objevit veřejné proměnné, dále proměnné chráněné, package a soukromé. [22]

Po deklaraci proměnných následují deklarace konstruktorů, třídních metod a instančních metod. Pořadí metod s ohledem na přístupnost se neaplikuje, metody by měly být vedle sebe, pokud spolu souvisí. [22] [5]

3.5.2.2 Konvenci pro práci v týmu

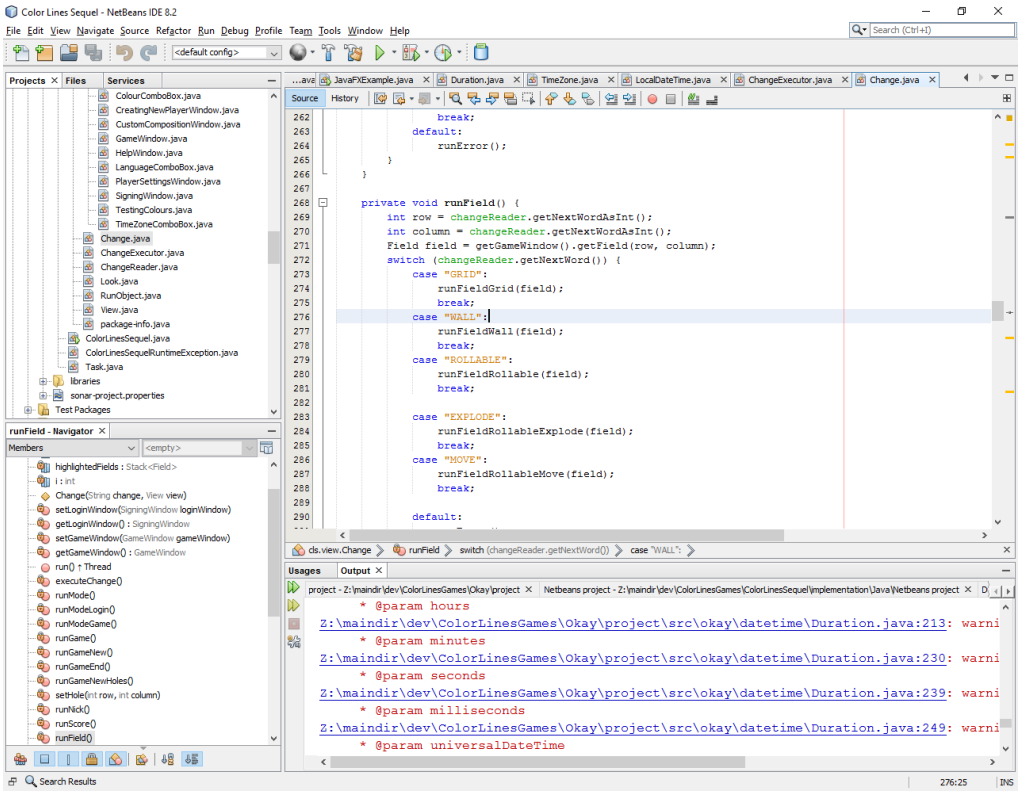
Během práce na skupinovém projektu by měly být stanoveny konvence pro psaní kódu. Konvence související s formátováním, například uvození bloku znakem „{„ umisťované na konec řádku, je možné nastavit v integrovaném vývojovém prostředí tak, aby se aplikovalo během automatického formátování. Konvence související s pojmenováním je nutné zajistit důsledným zvážením pojmenovávání. [5]

3.5.3 Refaktorování

Refaktorování je činnost provádění takových změn v kódu, které nezmění funkčnost, ale implementaci. Pro refaktorování je důležitá existence testů pro ověření, zda změnou implementace nebylo něco poškozeno a zda funkčnost zůstala stejná. Refaktorování by mělo pomáhat k větší srozumitelnosti a k snadnějšímu budoucímu rozvoji a změnám. Náklady vložené do refaktorování by měly mít proto v budoucnosti návratnost.
Integrovaná vývojová prostředí obsahují nástroje pro automatizování refaktorování některých refaktorovacích způsobů. Lze tak například snadno přejmenovávat. Refaktorování lze rozdělit na několik technik. Mezi tyto techniky patří úprava metod, přesouvání prvků mezi objekty, organizace dat, generalizace, zjednodušení volání metod, velké programování a jiné techniky. Refaktorování slouží například k odstranění duplicity, vytvoření nové třídy z části současné třídy, rozdělení zodpovědnosti metody vytvořením dalších metod nebo k nahrazení opakujících se stejných literálů konstantou. [5]

3.5.4 Integrovaná vývojová prostředí

Vyvíjet aplikace v Javě je možné i v primitivním textovém editoru, jakým je například Poznámkový blok v operačním systému Windows. Tento postup ale v sobě skýtá spoustu problémů, kterými jsou ochuzení o automatizování činností, přítomnost různých doplňků k práci, upozorňování na chyby v kódu.
Integrované vývojové prostředí je aplikace určená k vývoji a obsahující řadu funkcí, které zrychlí práci automatizováním a kontrolou současné podoby zdrojového kódu na správnost.
Funkčnost integrovaného vývojového prostředí lze rozšířit pomocí doplňků. Některé funkce jsou však již zabudované, například krokování a měření náročnosti aplikace na zdroje v Netbeans IDE.
Mezi příklady vývojových prostředí pro vývoj Java aplikací patří Netbeans IDE, Eclipse, IntelliJ IDEA a BlueJ. Eclipse a IntelliJ Idea mají největší podíl na trhu, Netbeans používá přibližně pouze 6% uživatelů. [18]



Obrázek 4: Vývojové prostředí Netbeans IDE

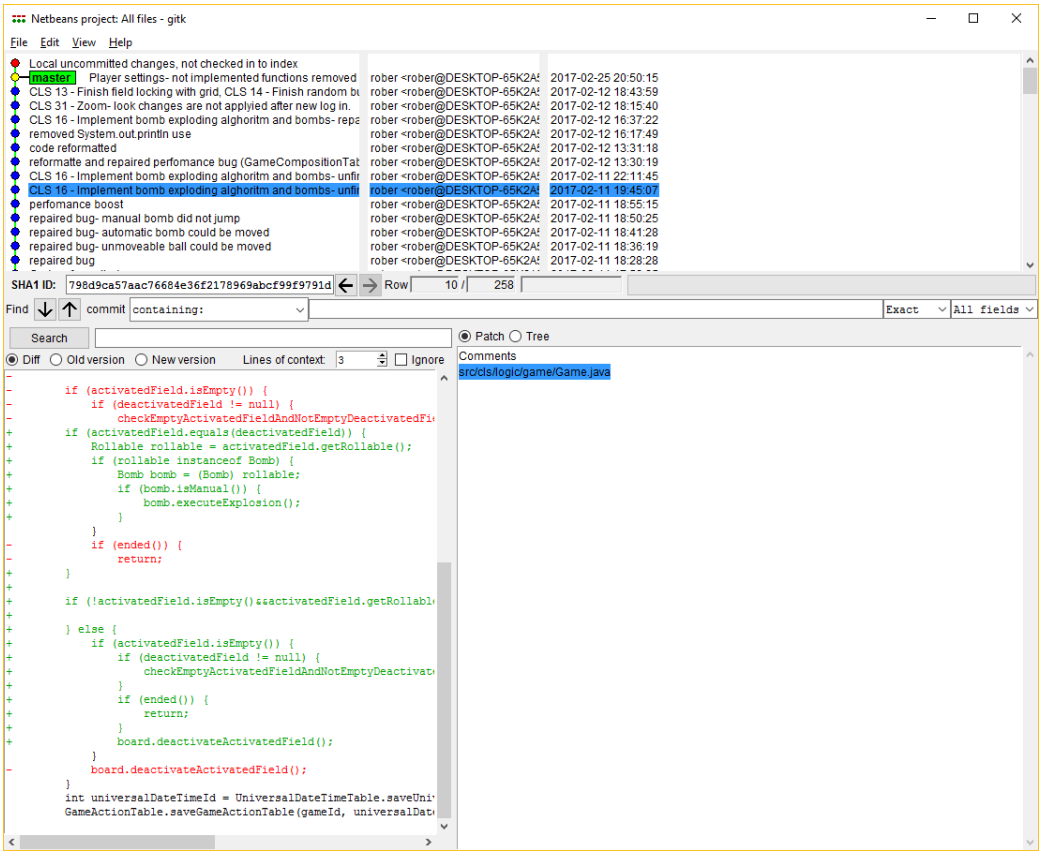
Zdroj: aplikace Netbeans IDE obsahující vlastní práci autora

3.5.5 Systémy pro správu verzí kódu

Systémy pro správu verzí kódu slouží k řízení změn ve zdrojových kódech umístěných v nějakých složkách. Je možné je rozdělit na centralizované a distribuované. U centralizovaných systémů je na serveru vždy pouze jedna podoba zdrojového kódu, programátor jako klient si stahuje aktuální podobu a po úpravě ji posílá zpět. U distribuovaných systémů, má programátor na svém počítači celou podobu systému pro správu verzí kódu a může se přepnout do jiného dřívějšího stavu i bez připojení se k serveru. Příkladem centralizovaného systému je Git a distribuovaného Svn. [19]

3.5.5.1 Git

S Gitem se komunikuje prostřednictvím textových příkazů, případně skrze vývojové integrované prostředí, které má grafickou nadstavbu nad těmito příkazy. Existují programy zobrazující vizuálně celý repozitář nějakého uložiště Gitu, například Gitk. Gitk umožnuje zobrazit všechny záchytné body uložení stavu, ukázat změny v kódu oproti předcházejícímu záchytnému stavu a provádět operace s repozitářem.

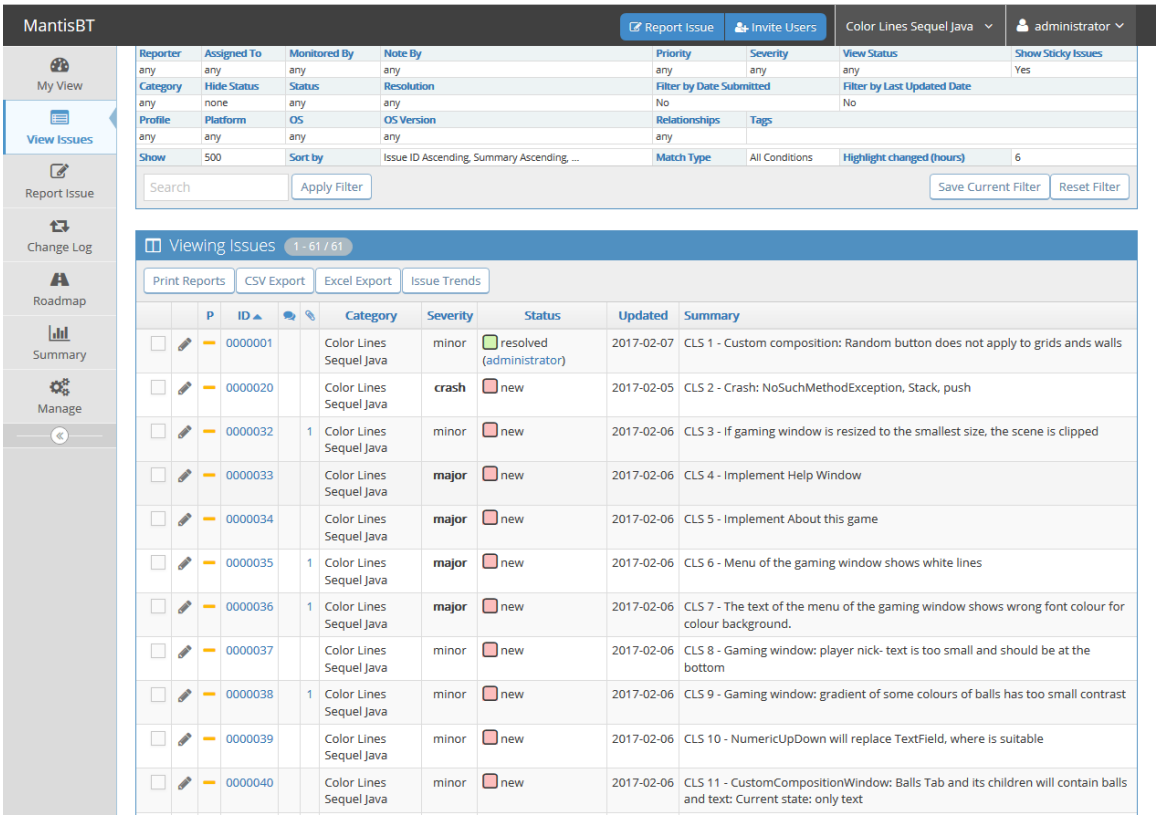


Obrázek 5: Aplikace Gitk

Zdroj: aplikace Gitk zobrazující verzovací údaje vytvořené autorem

3.5.6 Systémy pro správu problémů v projektu

Systém pro správu problémů v projektu je nějaká aplikace, často webová, která má uživatele a každý uživatel má nějakou roli. Rolemi mohou být programátor, zákazník, nebo uživatel. Tento systém spravuje databázi problémů nějakého většinou softwarového projektu. Tyto problémy mohou být chyba, vylepšení, změna. Problémy mohou mít další vlastnosti, například naléhavost, náročnost, status, výsledek rozhřešení, přiřazení programátorovi, název, shrnutí, čas vytvoření, přílohy. Příklady systémů pro správu problému jsou například BugZilla, YouTrack, BitBucket a MantisBT. [20] [21]



Obrázek 6: Aplikace MantisBT

Zdroj: aplikace MantisBT, data vložená autorem této práce

3.5.7 Testování

Testování je důležitým procesem ve vývoji softwaru. Pokud je software testován, znamená to, že lze prokázat jeho správnou funkčnost a snadněji provádět změny, protože lze ověřit nenarušení funkčnosti. To, že je software testován s úspěšnými výsledky, ale neprokazuje, že se v něm nevyskytují nějaké další chyby. Testování slouží ke zjišťování chyb nebo nedostatků. Každý test by měl dát výsledek prošel nebo neprošel. Jeden test by neměl obsahovat více než jeden testovací případ. Testy lze rozdělit na jednotkové, integrační, uživatelské, zátěžové, průzkumné bezpečnostní, podle scénářů a regresní.
Integrační testy slouží ke zjištění správné integrace modulů nebo systémů. Uživatelské testy testuje koncový uživatel nebo zákazník a jako výsledek je, zda aplikace splňuje očekávání. Testy podle scénářů jsou ruční testy, uživatel v grafickém uživatelském prostředí dle scénářů testuje, zda se aplikace chová dle očekávání. Regresní testy zjišťují, zda změny v kódu měly negativní neočekávaný vliv na jiná místa v kódu. [5] [6]

3.5.7.1 Jednotkové testy

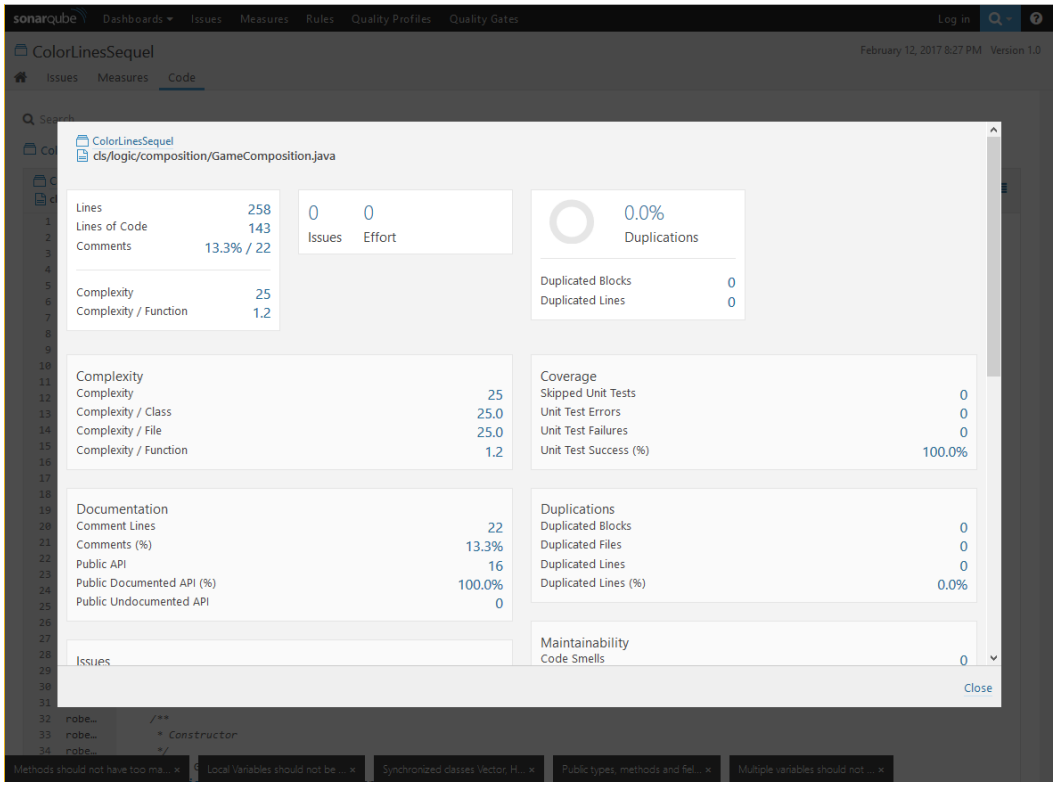
Testuje se na úrovni metod nebo tříd. Nejpoužívanější knihovnou v Javě pro jednotkové testování je JUnit. Dvěma základními typy testů je shodnost a přítomnost hození výjimky. U shodnosti se zkoumá, zda je návratová hodnota u volané metody shodná s očekáváním. U výjimek se zkoumá, zda dle očekávání výjimka padla nebo nepadla. [5] [6]

```
/**
    * Test of add method, of class JsonObject.
    */
```

```java
@Test
public void testAdd_ObjectIsLong() {
    //arrange
    JsonObject jsonObject = new JsonObject();
    JsonValueType expectedValue = JsonValueType.LONG;
    JsonValueType returnedValue;
    //act
    jsonObject.add("long", 2l);
    returnedValue = jsonObject.getJsonValueType("long");
    //assert
    assertEquals(expectedValue, returnedValue);
}
```

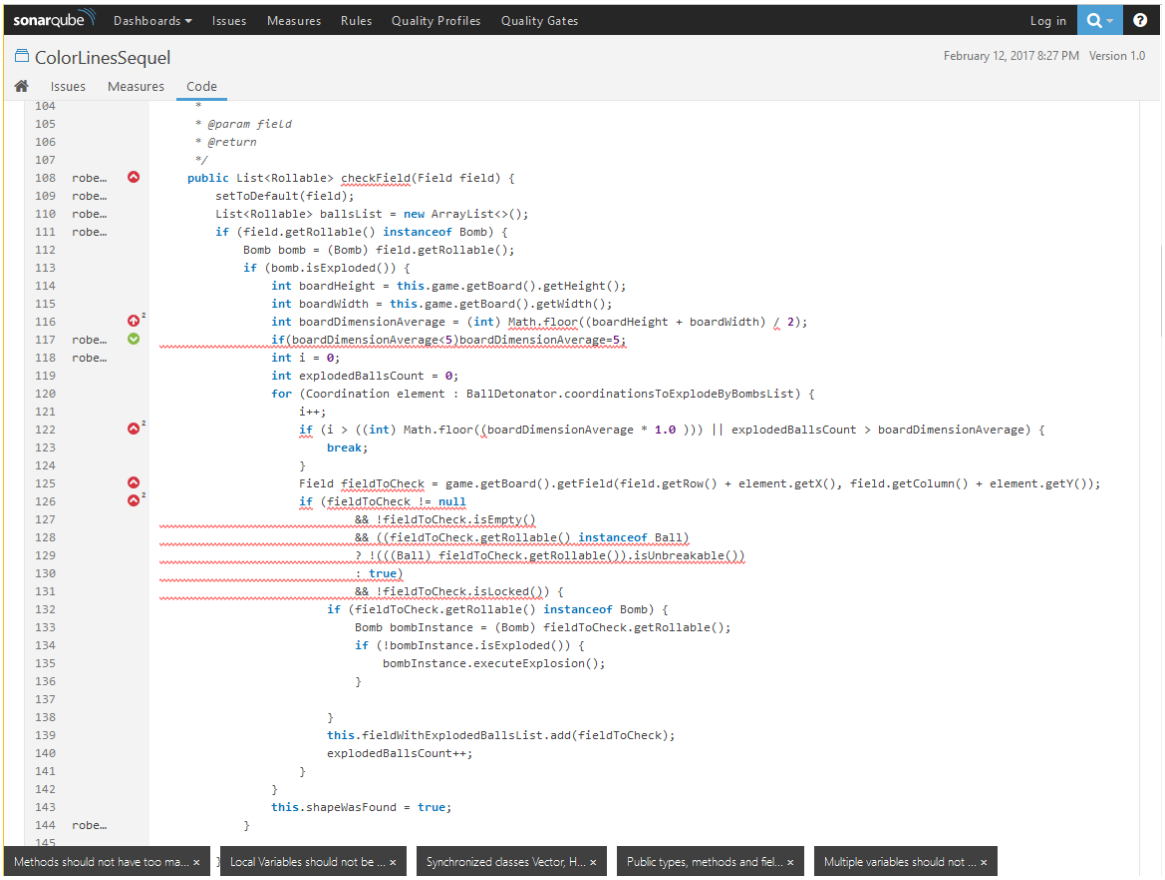
3.5.8 Statická analýza kódu

Statická analýza kódu je způsob pro posouzení správnosti kódu. Nástroj pro statickou analýzu kódu obsahuje chybové vzorce a následně je v kódu vyhledává. Mezi chybové vzorce patří bezpečnostní hrozby, výkonnostní díry, nedodržení jazykových konvencí, duplicita, příliš vysoká komplexita, absence dokumentace. Pomocí statické analýzy kódu nelze ověřit správnou funkčnost, ale pouze hledat chybové vzorce. Příkladem statického analyzátoru je PMD nebo SonarQube. [6]



Obrázek 7: Aplikace SonarQube- statistiky

Zdroj: aplikace SonarQube, data vygenerována z vlastní práce autora



Obrázek 8: Aplikace SonarQube- analýza kódu

Zdroj: aplikace SonarQube, data vygenerována z vlastní práce autora

3.6 Modely vývoje softwaru

Existují různé metody vývoje softwaru, všechny ale mají za cíl vytvoření softwarového produktu a zahrnují aktivity specifikace požadavků, návrh, implementaci, ověření softwaru a další rozvoj softwaru. Mezi hlavní modely patří vodopádový a iterativní model vývoje. [7]

3.6.1 Vodopádový model

Vodopádový model sestává z několika cyklů, z analýzy, návrhu, implementace, testování a údržby. Každý cyklus může začít až po skončení předcházejícího, což znamená nevýhodu v nemožnosti více cyklů v jeden okamžik a klade větší potřebu na důkladnost počáteční analýzy. Pokud je v některém cyklu zjištěna existence chyby v některém z cyklů předcházejících, nelze se vrátit a je nutné projít všemi zbývajícími cykly. V každém cyklu vznikají různé dokumenty, které mohou využívat vedoucí pracovníci k přehledu o aktuální stavu. Aplikace se uživatelům předává až v konečné podobě. [7]

3.6.2 Iterativní model

Iterativní vývoj spočívá v implementaci, která se opakovaně předkládá uživatelům aplikace ke kontrole a dle jejich komentářů se implementace upravuje. Výhodou je existence zpětné vazby. Analýza, vývoj a testování v iterativním modelu jsou prováděny

najednou a vzájemně se ovlivňují. Aplikace se uživatelům předkládá průběžně i přesto, že nejsou dosud implementovány všechny požadované funkce. [7]

4 Vlastní práce

Součástí této práce je i implementace logické hry ve formě desktopové aplikace v Javě a knihovna Okay. Knihovna Okay je pokryta kromě balíčku simplicity jednotkovými testy. Logická hra i knihovna Okay byly podrobeny statické analýze pomocí nástroje SonarQube. Tato statická analýze vedla k odstranění mnoha chybových vzorů. Byl využit systém pro správu verzí Git a systém pro správu problémů MantisBT.

4.1 Víceúčelová knihovna Okay v Javě

Okay je knihovna v Javě s balíčky pro různé účely. Vznikla během vývoje logické hry odštěpením několika balíčků obecného charakteru. Během oddělování těchto obecných balíčků vzniknul problém. Balíčky byly závislé na části kódu logické hry. Refaktorováním byla tato závislost odstraněna. Knihovna Okay se skládá ze šesti základních balíčků. Některé tyto balíčky jsou závislé na ostatních a nelze proto použít v nějakém projektu pouze některé balíčky. Je nutné připojit celou knihovnu Okay. Logická hra používala cizí knihovnu MinimalJson pro práci s Json objekty před vytvořením balíčku json.

4.1.1 Balíček collections

Balíček collections obsahuje implementaci datových struktur fronta, zásobník a slovník. Implementace těchto struktur využívá vnitřně datovou strukturu jednosměrný spojový seznam. Jednosměrný spojový seznam je seznam uzlů navzájem spolu propojených, každý uzel zná uzel následující.

4.1.1.1 Třída Stack

Třída Stack reprezentuje datovou strukturu zásobník. Do zásobníku se hodnoty ukládají na jeho vrchol, z vrcholu se také získávají. Třída Stack obsahuje metody pro uložení hodnoty na vrchol, získání hodnoty z vrcholu s odstraněním nebo bez odstranění, získání počtu prvků a zjištěný informace, zda je zásobník prázdný. V Javě lze třídy kolekcí implementující rozhraní Iterable procházet smyčkou for. Třída Stack rozhraní Iterable implementuje.

4.1.1.2 Příklad užití třídy Stack

```
Stack<String> stack = new Stack<>();
stack.push("John").push("Jack").push("Anne").push("Charlie");

System.out.println(stack.pop());

stack.push("Jane");

for(String element: stack)
{
System.out.println(element);
```

```
}
```

4.1.1.3 Metoda push

```
/**
     * Pushes new element.
     *
     * @param element
     * @return pushed element
     */
    public Stack<T> push(T element) {
        Node current = first;
        first = new Node();
        first.element = element;
        first.next = current;
        total++;
        return this;
    }
```

4.1.1.4 Třída Dictionary

Třída Dictionary používá dále třídy DictionaryNode a DictionaryKeyIterator. Třída Dictionary reprezentuje datovou strukturu mapa a umožňuje ukládat hodnoty adresované klíči.
DictionaryNode je uzel spojového seznamu specializovaný pro slovník. DictionaryKeyIterator slouží k procházení všech klíčů v slovníku. Třída Dictionary je vhodná pro postupné procházení, v případě náhodného přístupu dochází k poklesu výkonu.

4.1.2 Balíček database

Balíček database slouží k vytvoření abstrakce nad prací s relační databází pomocí rozhraní JDBC. Tato abstrakce vede k zjednodušenému používání, ale ovládání databáze pomocí dotazovacího jazyka SQL je zachováno. Balíček database používá vnitřně relační databázi SQLite, která pracuje nad souborem na disku. Skrytím implementace je ale skryto mnoho nepodstatného. Balíček database obsahuje třídy Database, DatabaseConnection, ResultOfSqlQuery a SqlCommandQueue.
Třída **Database** slouží k práci s databází jako celkem a obsahuje metody k vytvoření a odstranění databáze a dále k vytvoření instance třídy DatabaseConnection za pomoci názvu již existující databáze.
Třída **DatabaseConnection** je určena ke spouštění příkazů jazyka SQL pomocí několika metod dle toho, zda se něco vrací. Například metody execute spouští příkaz jazyka SQL a jako návratovou hodnotu vrací sloupec id naposledy přidaného řádku, což lze využít při vkládání nových řádků, kdy je hodnota sloupce id generována automaticky a je potřeba tuto hodnotu zjistit.
ResulOfSqlQuery reprezentuje výsledek dotazu o více řádcích a vnitřně využívá balíček json.
CommandQueue je fronta, do které lze uložit více SQL příkazů a poté je z důvodu úspory výkonu spustit pomocí třídy DatabaseConnection najednou.

4.1.2.1 Metoda getRow

Metoda getRow třídy DatabaseConnection má jako parametr název tabulky a číslo řádku a vrací reprezentaci tohoto řádku v podobě instance třídy JsonObject z balíčku json.

```
/**
     *
     * @param tableName
     * @param id
     * @return row with the given id from the given table as a json
object
     */
    public JsonObject getRow(String tableName, int id) {
        JsonObject row = new JsonObject();

        JsonArray columns = new JsonArray();

        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("select * from ");
        stringBuilder.append(tableName);
        stringBuilder.append(" where id=");
        stringBuilder.append(id);

        String command = stringBuilder.toString();

        setConnection();
        ResultSet resultSet = null;
        try {
            resultSet = statement.executeQuery(command);

            ResultSetMetaData rsmd = resultSet.getMetaData();
            int columnCount = rsmd.getColumnCount();
            for (int i = 1; i <= columnCount; i++) {
                String columnName = rsmd.getColumnName(i);
                columns.addString(columnName);
            }

            if (!resultSet.next()) {
                throw new OkayRuntimeException("There is no row with id "
+ id + " in table " + tableName);
            }

            for (int i = 0; i < columns.getCountOfItems(); i++) {
                String columnName = columns.getString(i);
                String value = resultSet.getString(columnName);
                row.addString(columnName, value);
            }

            connection.close();
        } catch (Exception e) {

Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE,
null, e);
            throw new IllegalStateException();
        } finally {

            try {
                if (resultSet != null) {
                    resultSet.close();
```

```
                }
                if (!connection.isClosed()) {
                    connection.close();
                }
            } catch (Exception e) {

Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE,
null, e);
            }
        }
        return row;
    }
```

4.1.2.2 Příklad použití balíčku database

```
Database.createDatabase("newDatabase");
        DatabaseConnection databaseConnection =
Database.createDatabaseConnection("newDatabase");

        SqlCommandQueue sqlCommandQueue = new SqlCommandQueue();
        sqlCommandQueue.add("create table customers(id integer,name
text,surname text,yearofbirth integer)");
        sqlCommandQueue.add("insert into customers
values(1,'John','Green',1954);");
        sqlCommandQueue.add("insert into customers
values(2,'Anne','Blue',1985);");
        sqlCommandQueue.add("insert into customers
values(3,'Peter','Orange',1990);");

        databaseConnection.executeMoreCommands(sqlCommandQueue);
        ResultOfSqlQuery resultOfSqlQuery =
databaseConnection.executeAndReturn("select * from customers");
        while (resultOfSqlQuery.hasNextRow()) {
            resultOfSqlQuery.moveToTheNextRow();
            System.out.println(
                    resultOfSqlQuery.getInt("id") + " "
                    + resultOfSqlQuery.getString("name") + " "
                    + resultOfSqlQuery.getString("surname") + " "
                    + resultOfSqlQuery.getString("yearofbirth")
            );
        }
```

Výpisem v konzoli je následující text:

1 John Green 1954
2 Anne Blue 1985
3 Peter Orange 1990

4.1.3 Balíček datetime

Účelem tohoto balíčku je vytvoření knihovny pro snadnější práci s časem, datem, časovými zónami a časovými úseky. Balíček datetime je inspirován balíčkem java.time ze standardní knihovny Javy pouze v názvech tříd a názvech některých metod. Třídy tohoto balíčku jsou neměnitelné objekty, kterým lze určit podobu pouze při vytvoření. Obsaženy jsou veřejné třídy Duration, LocalDate, LocalTime, LocalDateTime, UniversalDateTime, ZonedDateTime a TimeZone. Mezi pomocné třídy s přístupností package se řadí

DateUnitsValidator, TimeSource a TimeUnitsValidator. Některé metody těchto tříd používají vlastní implementaci algoritmů, složitější algoritmy jsou řešeny využitím balíčku java.time.

```
TimeZone oldTimeZone = new TimeZone("Europe/Prague");
ZonedDateTime oldZonedDateTime = new ZonedDateTime(new
LocalDateTime(1995, 11, 5, 10, 5, 4, 64), oldTimeZone);

TimeZone newTimeZone = new TimeZone("Australia/Sydney");
ZonedDateTime newZonedDateTime =
oldZonedDateTime.toZonedDateTime(newTimeZone);

System.out.println(newZonedDateTime.toString());
```

4.1.4 Balíček json

Balíček json obsahuje objektovou reprezentaci json objektu a json pole pomocí tříd JsonObject a JsonArray. Dále zde existuje výčet JsonValueType určující typ hodnot json objektu a json pole. Z důvodu pokrytí veškeré funkčnosti a snížení zodpovědnosti tříd balíček json obsahuje mnoho tříd viditelných pouze v balíčku json, například JsonArrayParser, JsonLong, JsonValue, JsonParser nebo JsonSpecialCharSequences. Třída JsonObject používá k ukládání hodnot třídu Dictionary a třída JsonArray implementuje pole pomocí kolekce ArrayList. Balíček json umožňuje vytvářet json objekty a pole z jejich textové reprezentace. Instance tříd JsonObject a JsonArray je možné převést na text pomocí metod toPrettyString nebo toMinimalString.

4.1.4.1 Příklad použití balíčku json

```
        JsonObject computer=new JsonObject();
        computer.addString("cpu", "Intel Corei7");
        computer.addInt("cores", 4);
        computer.addBoolean("desktop",false);
        computer.addObject("ram", new JsonObject("{\"capacity\":
8,\"units\":\"GB\"}"));
        JsonArray monitors=new JsonArray();
        JsonObject monitor1=new
JsonObject().addString("brand","Dell").addInt("width
px",1920).addInt("height px",1080);
        JsonObject monitor2=new
JsonObject().addString("brand","Samsung").addInt("width
px",1024).addInt("height px",768);
        monitors.addObject(monitor1);
        monitors.addObject(monitor2);
        computer.addArray("monitors",monitors);
        System.out.println(computer.toPrettyString());
```

Výpisem je následující text.

```
{
   "cpu":"Intel Corei7",
   "cores":4,
   "desktop":false,
   "ram":{
      "capacity":8,
```

```
        "units":"GB"
    },
    "monitors":[
        {
            "brand":"Dell",
            "width px":1920,
            "height px":1080
        },
        {
            "brand":"Samsung",
            "width px":1024,
            "height px":768
        }
    ]
}
```

4.1.5 Balíček pseudorandom

Balíček pseudorandom je určen ke generování pseudonáhodných čísel. Jelikož generovaná čísla jsou založena na nějakém výpočtu, nejedná se o čísla čistě náhodná. K získání čistě náhodných čísel nelze použít nějaký výpočet, ale lze použít například údaje získané sledováním nějakého přírodního jevu. Balíček pseudorandom obsahuje veřejnou třídu PseudoRandomGenerator a třídu Seed viditelnou pouze v balíčku.

4.1.5.1 Třída Seed

Třída Seed používá lineárně kongruentní funkci. Lineárně kongruentní funkce slouží k vygenerování řady čísel, které jeví známky náhodnosti. Lineární kongruentní funkce obsahuje tři konstanty, A, C a M, vztahy mezi těmito konstantami musí splňovat jisté podmínky a kombinace těchto konstant také určuje počet čísel ve vygenerované řadě, neboli její délku.

```
package okay.pseudorandom;

class Seed {

   private long currentSeed;
   private static final int A = 16807;
   private static final int C = 0;
   private static final long M = 2147483647;

   Seed(long seed) {
      this.currentSeed = seed;
   }

   long getNextNumber() {
      long number = ((A * currentSeed) + C) % M;
      this.currentSeed = number;
      return number;
```

```
    }

    void jump() {
        getNextNumber();
    }
}
```

4.1.5.2 Třída PseudoRandomGenerator

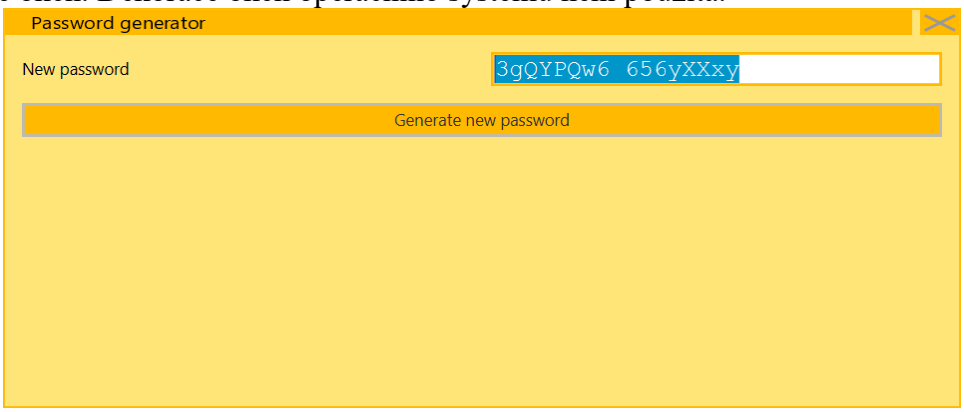
Třída PseudoRandomGenerator má tři instance třídy Seed. K vytvoření generátoru je potřeba nějaké číslo a datum s časem, což je nějaká instance třídy dědící od třídy DateTime. Generátor dle vstupních údaje předaných při vytvoření a pomocí příslušných metod vrací pseudonáhodná čísla. Při stejných vstupních údajích generátor generuje vždy stejnou řadu čísel. Jako vstupní údaj lze použít například aktuální čas a datum.

4.1.5.3 Příklad použití baličku pseudorandom

```
PseudoRandomGenerator pseudoRandomGenerator = new PseudoRandomGenerator(987,
new LocalDateTime(1991, 5, 24, 6, 28, 53, 862));

    for (int i = 1; i <= 400; i++) {
        System.out.println((pseudoRandomGenerator.getLong(0, 499999999999l));
    }
```

4.1.6 Balíček simplicity

Balíček simplicity je knihovna pro platformu JavaFX k tvorbě oken majících vlastní design dekorace oken. Dekorace oken operačního systému není použita.



Obrázek 9: Aplikace Password generátor využívající balíček simplicity knihovny Okay

Zdroj: vlastní práce autora

4.2 Logická hra Color Lines Sequel

4.2.1 Úvod

Color Lines Sequel je desktopová aplikace. K vytváření grafického uživatelského prostředí je využita platforma JavaFX. Trvalá data se ukládají do databáze SQLite. Je použita knihovna Okay. Princip této aplikace je inspirován hrou Color linez z roku 1995.

4.2.2 Pravidla hry

Hra má jednoho hráče, který má k dispozici hrací desku o nějakém počtu sloupců a řádek. Průsečíky sloupců a řádek obsahují pole. V náhodných polích se objevují míče. Cílem hráče je maximalizovat konečný výsledek počtu bodů. Body se získávají skládáním míčů stejné barvy do skupin nějakého tvaru, toho je dosaženo přesouváním míčů. Na začátku a v průběhu hry se objevují míče nové. Při každém vytvoření nových míčů se musí hráč rozhodnout, jaký míč kam přesunout. Pokud dojde k vytvoření skupiny míčů požadovaného tvaru, tyto míče zmizí a přičtou se další body, jinak se objeví v náhodných polích nové míče. Hra končí zaplněním všech polí míči.
Míč má barvu, hodnotu a může být nerozbitný nebo nepřesunutelný. Speciálním typem míčem jsou bomby. Pole může obsahovat z nějakého směru stěnu nebo je pole zamčené celé.

4.2.3 Přizpůsobení obtížnosti hry

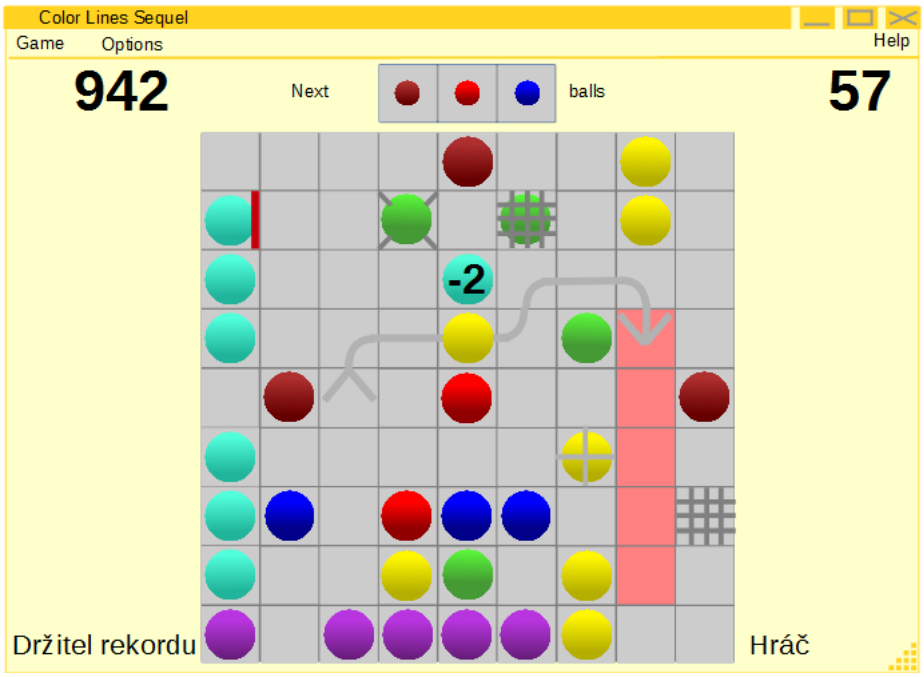
Obtížnost hry lze přizpůsobit skrze okno vyvolané v menu v položce Nová hra s vlastním sestavením. Uzpůsobit lze rozměry desky, frekvence barev míčů, počet objevujících se míčů nebo tvar skupiny míčů, jež se má skládat k získání dalších bodů. Tyto podmínky lze nastavit pouze před započetím hry a ne v jejím průběhu.

4.2.4 Přizpůsobení vzhledu aplikace

Je možné přizpůsobit barevný motiv nebo jazyk. U hrací desky lze uzpůsobit osvětlení míčů, dočasně zvýraznit pole, přes které se přesunul nějaký míč nebo zapnout zobrazení obrysu míčů.

4.2.5 Návrh uživatelského prostředí

K návrhu grafického uživatelského prostředí byla použita aplikace LibreOffice Draw. Design logické hry byl nejprve vytvořen v této aplikaci, až potom následoval vývoj v Javě.

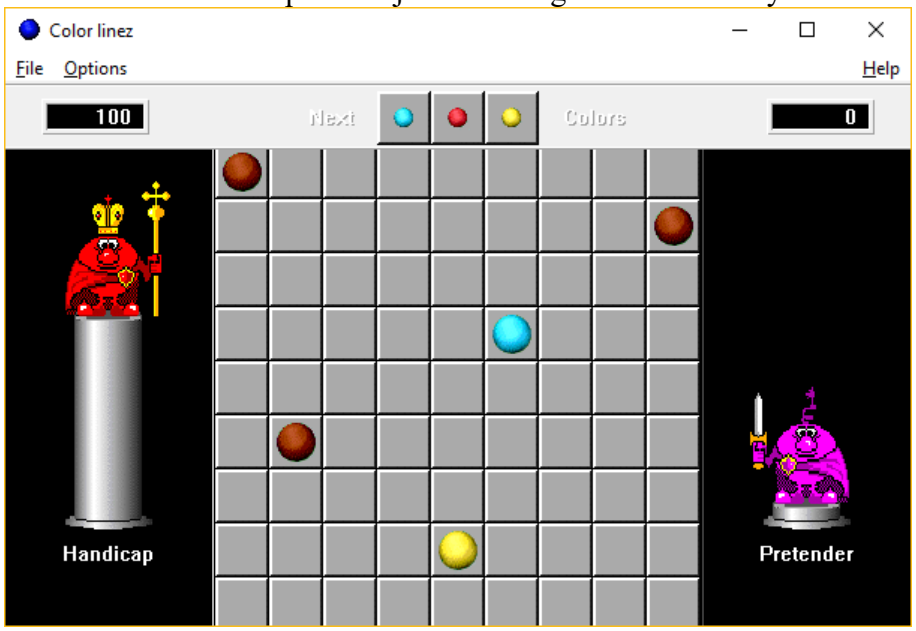


Obrázek 10: Grafický návrh aplikace Color Lines Sequel vytvořený v aplikaci Libre Office Draw

Zdroj: vlastní práce autora

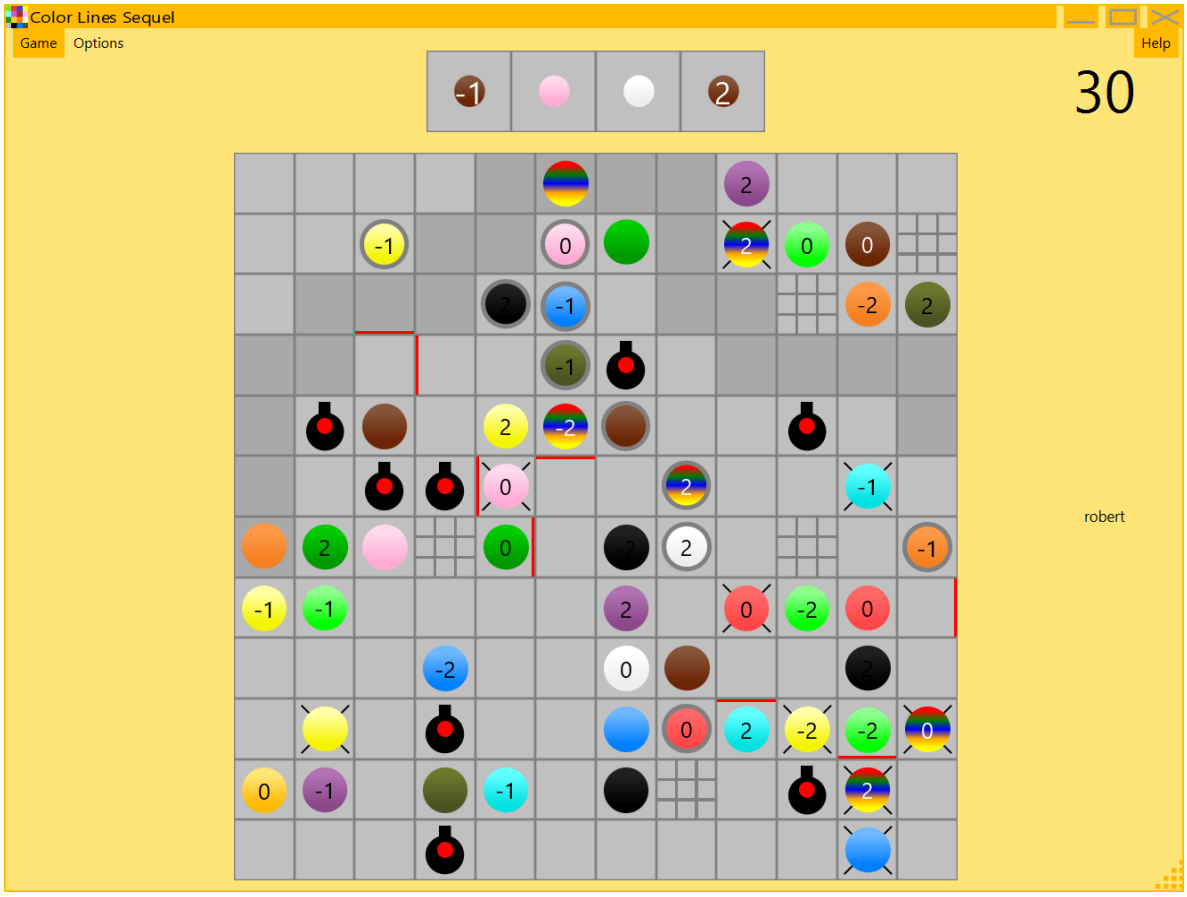
4.2.6 Porovnání vzhledu aplikací Color linez a Color Lines Sequel

Design aplikace Color Lines Sequel se na rozdíl od aplikace Color linez drží uměleckého směru miniminalismus a účelně používá jednoduché geometrické tvary.



Obrázek 11: Aplikace Color linez

Zdroj: aplikace Color linez, autoři Olga Demina, Igor Ivkin a Gennady Denisov



Obrázek 12: Aplikace Color Lines Sequel
Zdroj: vlastní práce autora

4.2.7 Architektura

Aplikace Color Lines Sequel je rozdělena do pěti vrstev, které jsou od nejnižších datová, doménová, logická, servisní a zobrazovací. Platí, že každá vrstva komunikuje pouze s vrstvou o úroveň níže. Výjimkou je ale například logická vrstva, které informuje zobrazovací vrstvu o změnách v herní desce.
Počet vrstev během vývoje postupně rostl. Nejprve byla přítomná pouze logická a zobrazovací vrstva, s růstem velikosti aplikace bylo vhodné rozdělit zodpovědnost do více vrstev.

4.2.7.1 Datová vrstva

Datová vrstva se skládá pouze z třídy Data. Smyslem datové vrstvy je zajištění relační databáze jako uložiště trvalých dat. Jako relační databázi datová vrstva používá minimalistickou SQLite, ale nekomunikuje s ní přímo pomocí rozhraní JDBC, nýbrž využívá balíček database knihovny Okay.

4.2.7.2 Doménová vrstva

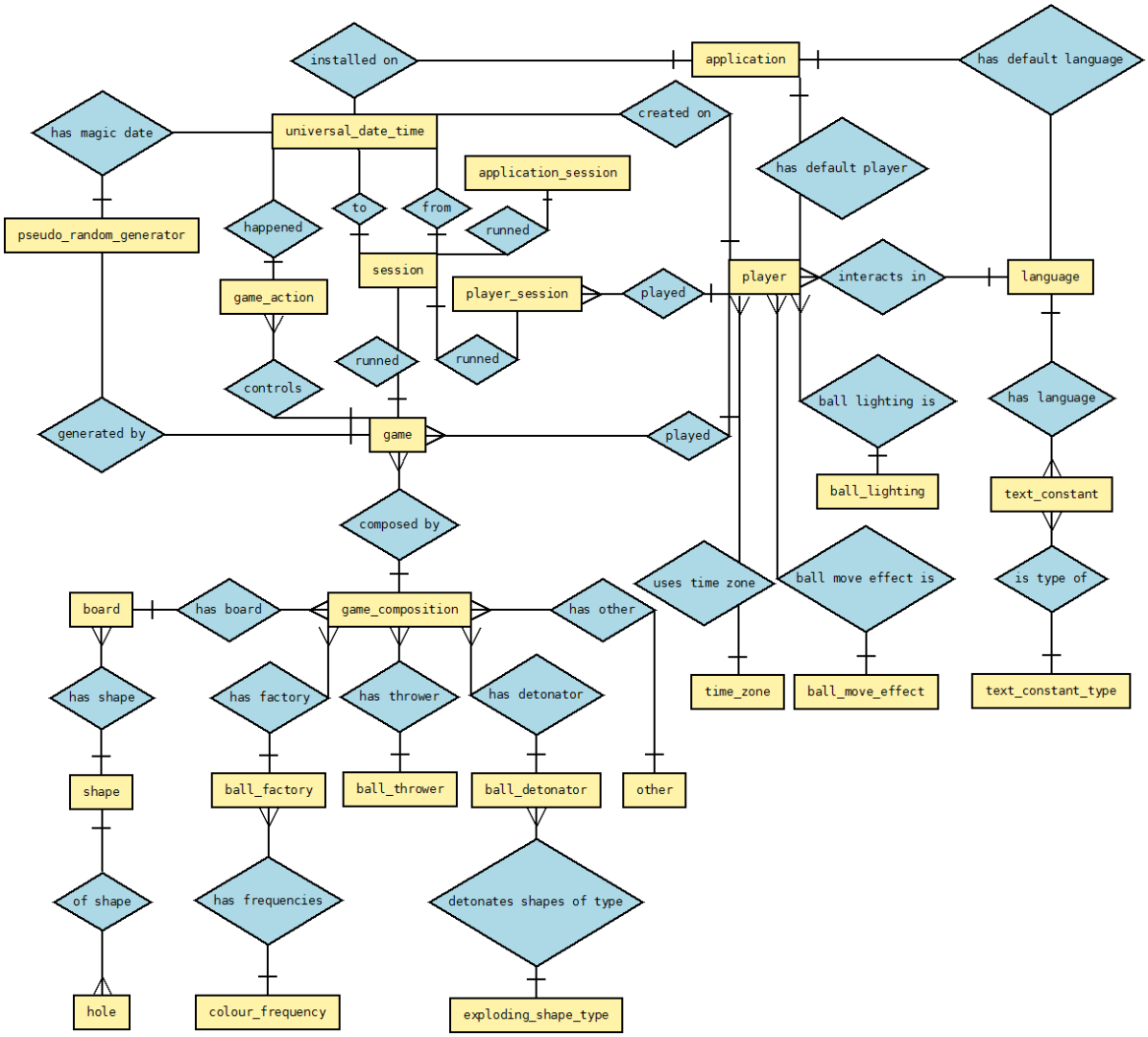
Doménová vrstva je určena ke správě a nastavení počátečních hodnot entit ukládaných do databáze prostřednictvím datové vrstvy. Těmito entitami se myslí tabulky.
Nastavení počátečních hodnot je prováděno při prvním spuštění aplikace.

Aplikace Color Lines Sequel při každém spuštění kontroluje, zda existuje soubor data.sqlite. Pokud tento soubor neexistuje, provede se automaticky instalace. Během instalace se vytvoří nová databáze reprezentovaná souborem data.sqlite, vytvoří se pomocí příkazů v souboru createtables.sql schéma databáze o 26 tabulkách a třída DomainInitializing pomocí statické metody install vloží do databáze počáteční hodnoty. Součástí instalace je také načtení jazykových konstant pro češtinu a angličtinu ze souborů cz.json a en.json do příslušných tabulek databáze.
Správa entit slouží k objektově relačnímu modelování. Objektově relační modelování je nástroj pro řešení objektovému přístupu k relačnímu modelu dat. Správa entit je zajištěna 26 třídami. Každá tato třída reprezentuje jednu tabulku a definuje skrze metody povolené operace na této tabulce. Například třída BoardTable obsahuje metody getBoardComposition, saveBoard, getId a další.
Následuje ukázka metody saveBoard:

```
public static int saveBoard(int gridProbability, int gridCount, int
wallProbaility, int wallCount, int shapeId) {
        StringBuilder insertCommandStringBuilder = new
StringBuilder("INSERT INTO board VALUES (null,");
        insertCommandStringBuilder.append(gridProbability).append(",");
        insertCommandStringBuilder.append(gridCount).append(",");
        insertCommandStringBuilder.append(wallProbaility).append(",");
        insertCommandStringBuilder.append(wallCount).append(",");
        insertCommandStringBuilder.append(shapeId).append(")");
        return
databaseConnection.execute(insertCommandStringBuilder.toString());
    }
```

4.2.7.3 ER diagram databáze

ER diagram neboli Entity relational diagram, slouží k modelování množiny typů entit a k definování vztahů mezi nimi.



Obrázek 13: ER diagram relačního modelu uložiště aplikace Color Lines Sequel

Zdroj: vlastní práce autora

4.2.7.4 Logická vrstva

Logická vrstva je reprezentovaná třídou Logic v balíčku cls.logic. Balíček cls.logic obsahuje dále administrativní třídy sloužící k přihlašování a odhlašování hráčů a zaznamenávání událostí.
Balíček cls.logic.composition obsahuje třídy reprezentující obtížnost hry, kde lze změnit například rozměry desky, například BallThrowerComposition. Ústřední třídou je ale třída GameComposition, která pomocí skládání obsahuje třídy podřízené. Instanci třídy GameComposition lze vytvořit i pomocí patřičné instance třídy JsonObject a opačně je možné reprezentaci třídy GameComposition převést na instanci třídy JsonObject. Třídu GameComposition lze pomocí metod zamknout proti změnám, kopírovat do nové instance nebo nastavit výchozí či náhodné hodnoty.

Balíček cls.logic.game obsahuje vlastní logiku hry. Neustále se opakuje smyčka aktivování pole a následného rozhodnutí logické vrstvy, co se stane.

4.2.7.5 Servisní vrstva

Servisní vrstva neobsahuje žádnou funkčnost, ale pouze vytváří rozhraní služeb pro vrstvu zobrazovací. Příkladem třídy z této vrstvy je třída AuthenticationService určená k vytváření nových hráčů, přihlašování a odhlašování.

4.2.7.6 Zobrazovací vrstva

Zobrazovací vrstva má na starost to, co se má zobrazit v grafické podobě uživateli. Využívá k tomu platformu JavaFX a balíček simplicity knihovny Okay. Zobrazovací vrstva obsahuje třídy reprezentující okna, herní prvky a třídy určené k řízení herní desky pomocí příkazů.

4.2.7.7 Řízení herní desky pomocí příkazů

Když hráč aktivuje kliknutím nějaké pole, pošle se tato informace logické vrstvě. Logická vrstva vyhodnotí, co se stane za akci. Pokud tato akce má vliv na to, co se zobrazuje uživateli, pošle logická vrstva zobrazovací vrstvě příkaz ke změně. Příkladem příkazu je „FIELD 4 9 ROLLABLE NEW BALL 10 2 UNMOVEABLE BREAKABLE“, který vytvoří v poli v řádku 4 a sloupci 9 žlutý nepřesunutelný míč s hodnotou dva.
Zobrazovací třída obsahuje k řízení těchto změn třídy ChangeExecutor, ChangeReader a Change. Třída ChangeReader rozdělí příkaz dle mezer do slov a postupně vrací tyto slova. Třída Change obsahuje logiku pro všechny akce ke změně herní desky a tato logika se vybírá dle konkrétního příkazu. Třída ChangeExecutor postupně vykonává příkazy ke změně podoby herní desky přicházející z logické vrstvy. Tyto příkazy jsou uchovávány ve frontě. Pomocí vláken je docíleno toho, že se některé příkazy začnou provádět až po dokončení posledního vykonávaného příkazu.

4.2.7.8 Syntaxe příkazů ke změně herní desky

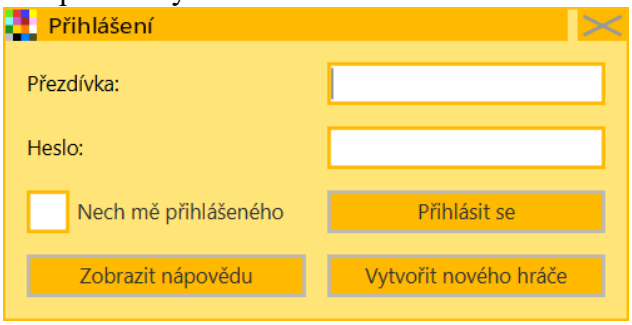
Aplikace Color Lines Sequel umožňuje vyzkoušet si provádění příkazů vyvoláním okna Execute command pomocí klávesy F12. Tato funkce ale slouží pouze k testování a může vyvolat pád aplikace Color Lines Sequel, pokud se klikne do nějakého pole.
Následující seznam zobrazuje syntaxi příkazů ke změně herní desky. Tyto příkazy jsou během hraní automaticky generovány.

```
MODE LOGIN|GAME
GAME NEW 9 9 3 HOLES
NICK|SCORE PLAYER|RECORD-HOLDER robert|120
FIELD 4 9 GRID ON|OFF
FIELD 4 9 WALL TOP|RIGHT|BOTTOM|LEFT ON|OFF
FIELD 4 9|NEXT 2 ROLLABLE NEW BALL 1-16|0 VALUE-2
UNMOVEABLE|MOVEABLE UNBREAKABLE|BREAKABLE
FIELD 4 9|NEXT 2 ROLLABLE NEW BOMB AUTOMATIC|MANUAL
FIELD 4 9 ROLLABLE INFLATE
FIELD 4 9 ROLLABLE JUMPING ON|OFF
FIELD 4 9 ROLLABLE EXPLODE
FIELD 4 9 ROLLABLE MOVE TO 4 8 TO 4 7 TO 4 6 TO 4 5 TO 4 4
NEXT 2 CLEAR
WAIT 4000
```

4.2.8 Ovládání aplikace Color Lines Sequel

4.2.8.1 Administrativa

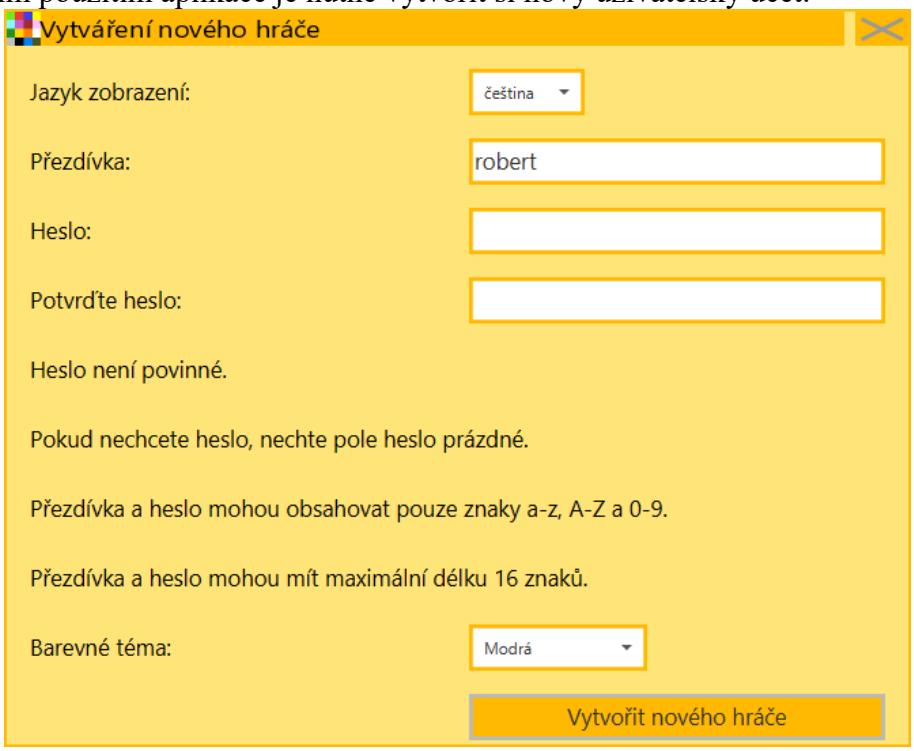
Používání aplikace Color Lines Sequel je podmíněno úspěšným přihlášením. Přihlášení probíhá zadáním správné přezdívky a hesla.



Obrázek 14: Přihlašovací okno

Zdroj: vlastní práce autora

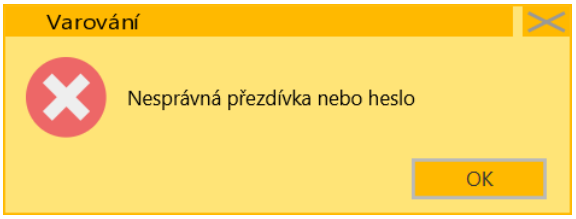
Před prvním použitím aplikace je nutné vytvořit si nový uživatelský účet.



Obrázek 15: Okno pro vytvoření nového hráče

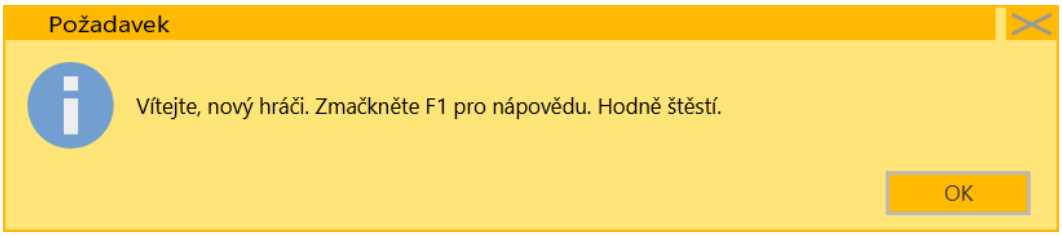
Zdroj: vlastní práce autora

Pokud jsou zadány nesprávné údaje pro přihlášení nebo vytvoření nového hráče, zobrazí se chybová hláška. Po vytvoření nového hráče se ukáže informativní okno.



Obrázek 16: Chybová hláška

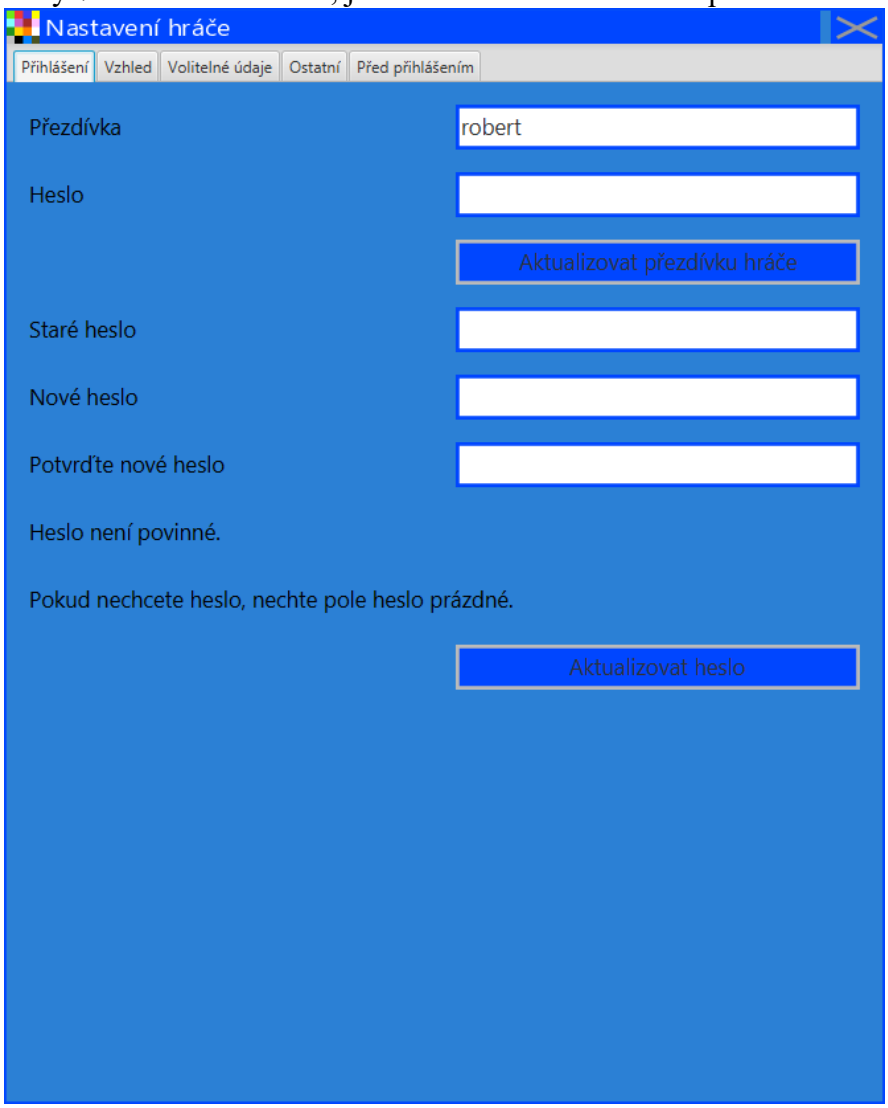
Zdroj: vlastní práce autora



Obrázek 17: Informativní hláška

Zdroj: vlastní práce autora

Ke každému uživatelskému účtu se váže řada nastavení a informací, která je možné změnit pomocí okna Nastavení hráče. Je možné změnit přihlašovací údaje, vzhled aplikace a herní desky, jazyk aplikace, osobní údaje o hráči a vzhled aplikace před přihlášení. Aby se projevily změny v nastavení vzhledu, je nutné se odhlásit a znovu přihlásit.

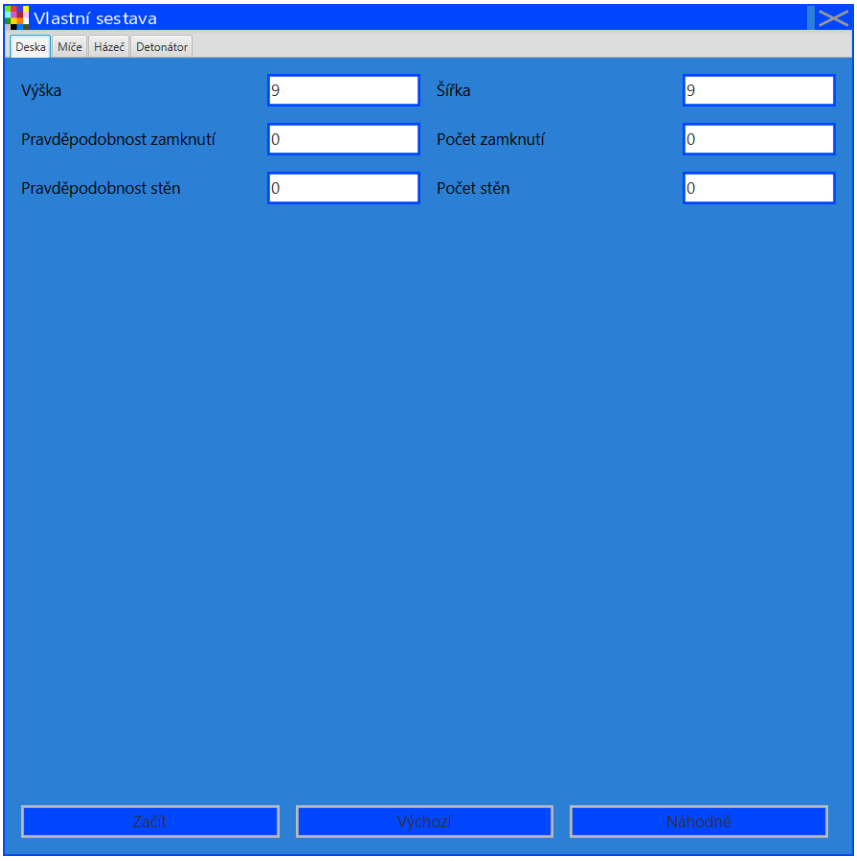


Obrázek 18: Okno pro nastavení hráče

Zdroj: vlastní práce autora

4.2.8.2 Uzpůsobení herní obtížnosti

Skrze položku v menu Nová hra s vlastním sestavením se spustí nové okno určené k uzpůsobení obtížnosti hry. Data do tohoto okna se načítají z logické vrstvy prostřednictvím instance třídy JsonObject a jsou reprezentována ve formátu json. Při aktivování tlačítka Začít se aktuální data v okně pošlou logické vrstvě a ta následně spustí novou hru s danou obtížností.



Obrázek 19: Okno pro uzpůsobení herní obtížnosti

Zdroj: vlastní práce autora

5 Zhodnocení výsledků praktické části

Hlavním cílem praktické části bylo vytvoření desktopové logické hry v Javě a víceúčelové knihovny.
Logická hra má následující funkčnosti:

- hráči se přihlašují prostřednictvím uživatelského účtu a hesla
- každý uživatelský účet má k sobě vázané různé informace o uživateli
- všechna trvalá data se ukládají do relační databáze SQLite
- aplikace umožňuje volbu jazyka uživatelského rozhraní
- aplikace umožňuje uzpůsobení vzhledu
- obtížnost logické hry lze uzpůsobit
- nafouknutí, skákání a přesun míče je animován
- přihlašování a odhlašování uživatele, zapnutí a vypnutí aplikace a každé kliknutí do pole je ukládáno jako událost do databáze i s otiskem času s přesností na milisekundy.

Víceúčelová knihovna se skládá z šesti hlavních balíčků pro následující funkce:

- implementace kolekcí mapa, fronta a zásobník
- generátor pseudonáhodných čísel
- práce s datem a časem
- relační databáze
- práce s formátem json
- vytváření oken používajících minimalistický design

Desktopová aplikace Color Lines Sequel je plně funkční a byla otestována autorem této bakalářské práce pomocí ručních testů.
Víceúčelová knihovna nebyla od začátku vývoje samostatným projektem, ale byla součástí desktopové aplikace. V průběhu vývoje desktopové aplikace ale došlo k izolaci víceúčelového kódu do knihovny.
Desktopová aplikace zprvu sestávala pouze ze dvou vrstev, logické a zobrazovací. S postupným růstem aplikace došlo k postupnému rozdělování zodpovědností do více vrstev.
V průběhu vývoje desktopové aplikace nastalo několik problémů. Bylo zjištěno, že pokud uživatel klikne do pole příliš brzy, začne se provádět další kód ještě před skončením minulého. Tato situace způsobovala pád aplikace. Řešením bylo použít vlákna k řízení běhu souběžného kódu a frontu, do které se budou ukládat instrukce identifikující aktivované pole. Další instrukce se začne provádět vždy až po skončení současné instrukce. Dalším problémem byla situace, kdy se začala v hrací desce provádět nějaká animace ještě před skončením animace současné. Pokud tyto animace byly v konfliktu, desktopová aplikace se začala chovat neočekávaně nebo padala. Řešením bylo opět použití vláken.
Během vývoje bylo také zjištěno, že desktopová aplikace zabírá extrémní množství paměti RAM při svém spuštění. Použití profileru vyřešilo tento problém zjištěním, že kvůli špatně sestavenému SQL dotazu dochází k vytváření miliónů nepotřebných instancí třídy JsonObject.

6 Závěr

V teoretické části byl charakterizován programovací jazyk Java. Byly popsány různé nástroje řešící problémy při vývoji.
V praktické části byla úspěšně vyvinuta logická hra Color Lines Sequel. Tato desktopová aplikace ukládá trvalá data do relační databáze a grafické uživatelské rozhraní je vytvořeno pomocí platformy JavaFX. Veškeré plánované funkce administrativy desktopové aplikace i samotné logické hry byly úspěšně implementovány a ručně otestovány.
V praktické části byla také úspěšně vytvořena knihovna Okay, která umožňuje pracovat s časem a datem, generovat pseudonáhodná čísla, pracovat jednoduše s relační databází, pracovat s formátem json a vytvářet okna mající svůj vlastní originální design.
Kód desktopové aplikace a knihovny dohromady čítá 12 664 řádek kódu a 209 souborů formátu java.
Knihovnu Okay lze dále rozšiřovat a vylepšovat. Balíčku json chybí logika pro validaci json objektů. V balíčku database lze více využít json objekty k reprezentaci řádků tabulek nebo umožnit práci s relační databází pomocí objektového paradigmatu místo SQL dotazů. Balíčku simplicity chybí implementace některých důležitých ovládacích prvků.
Desktopová logická hra v současné podobě ukládá mnoho informací do databáze, ale neexistuje k nim kód, který by je uměl interpretovat uživateli v grafické podobě. Tato interpretace by byla prostřednictvím tabulek a grafů.
U každé odehrané hry se ukládá její generátor pseudonáhodných čísel a všechna kliknutí do pole. Pomocí těchto informací je možné zpětně zrekonstruovat odehrané hry k jejich následné analýze. Je také možné vytvořit přehrávač odehraných her podobný přehrávači videí.
V logické hře není implantováno několik neplánovaných funkcí, například stanovení tvaru herní desky, možnost stanovení vlastního tvaru skupiny skládaných míčů, ukládání rozehraných her nebo jejich načítání.
Hráč během hraní hry neustále volí svoji strategii. Dalším možným rozšířením by byla automatická analýza aktuálního stavu hry, která by dokázala rozhodnout se dále, jak hrát a tímto způsobem hráči radit nebo zhodnotit jeho odehranou hru.
Uživatelské prostředí desktopové aplikace je sice možné nastavit pro češtinu nebo angličtinu, rozšířením by ale bylo možnost nastavit i jiné jazyky, či vytvořit si vlastní novou jazykovou mutaci.
Desktopová aplikace na rozdíl od knihovny nebyla podrobena jednotkovým testům a obsahuje řadu chybových vzorů detekovaných pomocí statické analýzy kódu.
Knihovnu i desktopovou aplikaci je možné dále refaktorovat k přehlednějším kódu.

7 Seznam použitých zdrojů

1. JAMES KEOGH. *Java bez předchozích znalostí*. Brno, Computer Press, 2012. 274 s. ISBN 978-80-251-0839-0
2. HERBERT SCHILDT. *Mistrovství - Java: Kompletní průvodce vývojáře*. Brno, Computer Press, 2014. 1224 s. ISBN 978-80-251-4145-8
3. KISHORI SHARAN. *Learn JavaFX*. New York, Apress, 2015. 1190 s. ISBN 978-1-4842-1143-4
4. PECINOVSKÝ. *Návrhové vzory*. Brno, Computer Press, 2007. 527 s. ISBN 978-80-251-1582-4
5. ROBERT C. MARTIN. *Čistý kód*. Brno, Computer Press, 2009. 423 s. ISBN 978-80-251-2285-3
6. PAVEL HEROUT. *Testování pro programátory*. České Budějovice, Kopp, 2016. 405 s. ISBN 978-80-7232-481-1
7. IAN SOMMERVILLE. *Softwarové inženýrství*. Brno, Computer Press, 2013. 680 s. ISBN 978-80-251-3826-7
8. LUBOSLAV LACKO. *1001 tipů a triků pro SQL*. Brno, Computer Press, 2011. 415 s. ISBN 978-80-251-3010-0
9. TAPIKI. *5 Features in Java 9 that WILL Change How You Develop Software (and 2 That Won't)*. [online]. [cit. 2017-02-14]. <http://blog.takipi.com/5-features-in-java-9-that-will-change-how-you-develop-software-and-2-that-wont/>.
10. JSON.ORG. *JSON*. [online]. [cit. 2016-09-26]. <http://www.json.org/>.
11. GITHUB. *GitHub - ralfstx/minimal-json: A fast and small JSON parser and writer for Java*. [online]. [cit. 2017-02-26]. <https://github.com/ralfstx/minimal-json>.
12. ORACLE. *Java API for JSON Processing*. [online]. [cit. 2017-02-21]. <http://www.oracle.com/technetwork/articles/java/json-1973242.html>.
13. SQLITE.ORG. *About SQLite*. [online]. [cit. 2016-10-01]. <https://www.sqlite.org/about.html>.
14. GLUONHQ. *JavaFXPorts*. [online]. [cit. 2016-12-15]. <http://docs.gluonhq.com/javafxports/>.
15. GOOGLE PLAY. *Ensemble8 – Android Apps on Google Play*. [online]. [cit. 2016-12-14]. <https://play.google.com/store/apps/details?id=org.javafxports.ensemble8>.
16. GITHUB. *GitHub - kripken/sql.js: SQLite compiled to JavaScript through Emscripten*. [online]. [cit. 2017-02-27]. <https://github.com/kripken/sql.js>.
17. GITHUB. *GitHub - SQLDroid/SQLDroid: SQLite JDBC driver for Android & now also for non-Android platforms*. [online]. [cit. 2017-02-27]. <https://github.com/SQLDroid/SQLDroid>.
18. BAELDUNG. *The Market Share of Java IDEs in Q2 2016 | Baeldung*. [online]. [cit. 2016-11-23]. <http://www.baeldung.com/java-ides-2016>.
19. ABCLINUXU. *Distribuované verzovací systémy – úvod (1)*. [online]. [cit. 2017-01-24]. <http://www.abclinuxu.cz/clanky/distribuovane-verzovaci-systemy-uvod-1>.
20. TECHOPEDIA. *What is Bug Tracking? - Definition from Techopedia*. [online]. [cit. 2017-02-20]. <https://www.techopedia.com/definition/25910/bug-tracking>.

21. HAREESH B., PRAVEEN UDUPA. *Bug Tracking System*. [online]. [cit. 2017-02-2]. <https://www.slideshare.net/kishankishanacharya/bug-tracking-system-38586218>.
22. ORACLE. *Code Conventions for the Java Programming Language: 9. Naming Conventions*. [online]. [cit. 2017-03-05]. <http://www.oracle.com/technetwork/java/codeconventions-135099.html>
23. TIOBE. *TIOBE Index | TIOBE - The Software Quality Company*. [online]. [cit. 2017-01-22]. <http://www.tiobe.com/tiobe-index/>

8 Přílohy

1. Soubor s aplikací Color Lines Sequel - desktopovaaplikace.zip
2. Soubor s knihovnou Okay - knihovna.zip
3. Soubor s ER diagramem - er.dia
4. Soubor s grafickým návrhem desktopové aplikace - Designing ui.odg
5. Soubor s aplikací pro generování hesel - PasswordGenerator.zip