Vyšší odborná škola
a Střední průmyslová škola elektrotechnická
Plzeň, Koterovská 85

DLOUHODOBÁ MATURITNÍ PRÁCE S OBHAJOBOU

Téma: **Vektorový Editor**

Autor práce: **Michal Vojtuš**
Třída: **4. I**
Vedoucí práce: **Mgr. Karel Vlna**
Dne: **30. 4. 2022**

Hodnocení:

Vyšší odborná škola

a Střední průmyslová škola elektrotechnická

Plzeň, Koterovská 85

Zadání dlouhodobé maturitní práce

Žák:	**Michal Vojtuš**
Třída:	**4. I**
Studijní obor:	**18-20-M/01 Informační technologie**
Zaměření:	**Vývoj aplikací a správa systémů**
Školní rok:	**2021 – 2022**
Téma práce:	***Vektorový editor***

Pokyny k obsahu a rozsahu práce:

Vytvořit grafický vektorový editor pomocí jazyka Java, který umožní vykreslení základních tvarů. A to elipsa, kruh, obdélník, čtverec, čára, lomená čára, trojúhelník či další tvary. Každý tvar bude možné vytvořit, editovat a smazat. Úpravou tvaru je myšlena změna typu a barvy čáry, změna výplně, změna velikosti a umístění. Výsledný obrázek bude možné exportovat do bitmapového souboru. Práce bude vypracována v týmu dvou osob.

Dílčí části práce:

1. Stanovit pravidla a postup týmové spolupráce.
2. Charakterizovat platformu Java.
3. Zvolit vhodné vývojové prostředí a charakterizovat jej.
4. Navrhnout vhodné grafické uživatelské prostředí editoru.
5. Realizovat jednotlivé funkce grafického editoru.
    a. Kreslení základních tvarů
    b. Změna barvy, tloušťky a typu čáry.
    c. Změna výplně tvaru.
    d. Změna velikosti a umístění.
    e. Odstranění tvaru.
    f. Export do bitmapového souboru.
6. Vytvořit ukázkové skici.
7. Vypracovat dokumentaci maturitní práce dle Přílohy 7i vnitřního předpisu školy.

Určení částí tématu zpracovávaných jednotlivými žáky:

Michal Vojtuš

- 1. Stanovit pravidla a postup týmové spolupráce.
- 2. Charakterizovat platformu Java.
- 4. Navrhnout vhodné grafické uživatelské prostředí editoru.
- 5. Realizovat jednotlivé funkce grafického editoru část d, e, f.
- 6. Vytvořit ukázkové skici.
- 7. Vypracovat dokumentaci maturitní práce dle Přílohy 7i vnitřního předpisu školy.

Ondřej Pták

- 1. Stanovit pravidla a postup týmové spolupráce.
- 3. Zvolit vhodné vývojové prostředí a charakterizovat jej.
- 5. Realizovat jednotlivé funkce grafického editoru část a, b, c.
- 6. Vytvořit ukázkové skici.
- 7. Vypracovat dokumentaci maturitní práce dle Přílohy 7i vnitřního předpisu školy.

Požadavek na počet vyhotovení maturitní práce: *2 výtisky*

Termín odevzdání: ***22. dubna 2022***

Čas obhajoby: ***15 minut***

Vedoucí práce: **Mgr. Karel Vlna**

Projednáno v **katedře VTT** a schváleno ředitelkou školy.

V Plzni dne: 30. října 2021

Ing. Naděžda Mauleová, MBA, v.r.

ředitelka školy

Anotace

Úkolem tohoto projektu bylo vytvoření vektorového editoru, grafický program, který je schopen vytvářet obecné tvary jakožto čtverce, obdélníky, kruhy a další. Tyto tvary je možné změnit, tím je myšlena změna jejich barvy výplně, ohraničení, smazat, posouvat nebo také editovat. Následně by bylo možné celé plátno, na němž se všechny tvary nachází, exportovat jakožto obrázek.

„Prohlašuji, že jsem tuto práci vypracoval samostatně a použil(a) literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací.“
„Souhlasím s využitím mé práce učiteli VOŠ a SPŠE Plzeň k výuce.“

V Plzni dne: …..... Podpis: ….....

Obsah

1 Úvod

Téma vektorový editor jsme si vybrali, protože nám přišlo jakožto nejzajímavější v kategorii Vývoj aplikací a jeden z nás už má zkušenosti s vytvářením kreslícího programu z předchozích let. Konceptem naší práce je vytvořit vektorový editor pomocí programovacího jazyku Java. Jakožto správný vektorový editor by měl být program schopen kreslit základní tvary například kruh, elipsu, čtverec obdélník a jiné. Následně je upravovat na přání uživatele, úpravou je myšlen přesun tvaru po plátně, změna jeho barvy okrajů a výplně, zvětšení či zmenšení daného tvaru nebo možné odstranění tvaru, a nakonec výsledný obrázek exportovat do bitmapového souboru. Naše praktické provedení bude určeno naším konečným výběrem mezi frameworky Swing a Java FX Našeho cíle chceme dosáhnout použitím volně dostupných technologií jakožto školní Gitlab a samotný program stavět ve vývojovém prostředí IntelliJ. Naším cílem není nejen splnění zadání, ale hlavně vytvořit uživatelsky přívětivou a stabilní aplikaci. Nejdříve si musíme osvojit náš programoví jazyk společně s vývojovým prostředím, vytvořit uživatelské rozhraní, které bude v symbióze pracovat s funkcemi programu.

2 Stanovení pravidel a postupu týmové práce

2.1 Spolupráce

Náš dvoučlenný tým tvoří žáci Ondřej Pták a Michal Vojtuš. Shodli jsme se, že pracovat na našem projektu budeme každý týden v pátek během hodin MTP, přičemž řešení detailů či pokračování v krocích, které jsme nestihli, budeme řešit flexibilně během zbytku týdne nebo na další hodině. Konzultace ohledně maturitní práce probíhají během víkendu. Shodli jsme se, že projekt budeme realizovat ve vývojovém prostředí IntelliJ pomocí jazyku Java – jak bylo řečeno v zadání a že naším úložištěm bude náš školní GitLab projekt. Informace pro tvorbu kódu hledáme a části kódu píšeme nezávisle na sobě, pokud je to možné. Při společných konzultacích naše práce spojujeme či vylepšujeme a opravujeme. Popřípadě pokud si jeden z členů neví rady, tak mu druhý člen s problémem pomůže. Naše části práce se budou často prolínat a budeme na nich pracovat společně. Párkrát jsme museli dělat práce toho druhého, protože bylo častěji jednodušší udělat celou část programu sám než jen polovinu, kterou by pak musel vysvětlovat kolegovi, který by si ji musel osvojit, aby na ní mohl efektivně pokračovat. Nejčastěji se toto stávalo při vývoji 5. bodu zadání – vytváření tvarů a jejich editace. Bylo mnohem jednodušší se domluvit, jak tento problém vyřešíme, společně vytvořit kostru programu na kterou jsme pak mohli rychle a nezávisle na sobě měnit, kopírovat a splnit tento bod zadání viz 5.1. kreslení základních tvarů.

3 Charakterizace platformy Java

3.1 Základní definice

Platforma Java je sada programů od firmy Sun Microsystems, umožňující vývoj a spuštění programů v programovacím jazyku Java. Později se však společnost Sun Microsystems sloučila s Oracle corporation, která dnes vlastní platformu Java i programovací jazyk Java. Pomocí virtualizace je kompilátor se sadou knihoven realizován pro různé operační systémy a hardware, aby programy v jazyce Java mohly běžet nezávisle na operačním systému, nebo procesoru.

3.2 Historie Javy

Vše začalo v roce 1990 již zmíněnou firmou Sun Microsystems, kde platforma Java a Java jakožto programovací jazyk měli představovat alternativu pro programovací jazyky C++/C. Na vývoji začal pracovat softwarový vývojář Patrick Naughton, který nebyl spokojen se stavem rozhraní a nástrojů C a C++, které Sun Microsystems používal. Mezitím co přemýšlel nad přesunem do firmy NeXT, dostal nabídku pracovat na nové technologii Stealth Project. Následující rok se Naughton přidal k softwarovým programátorům Jamesovi Goslingovi a Miku Sheridanovi a Stealth Project byl přejmenován na Greentalk a začal používat příponu .gt. Následně byl přejmenován na Oak a byl vyvíjen jakožto součást Green project. Jméno Oak bylo zvoleno, protože dub představoval symbol síly a je zvolen jako národní strom mnoha zemí, jako jsou USA, Francie, Německo nebo Rumunsko. V roce 1995 byl Oak přejmenován na Java, protože to již byla ochranná známka společnosti Oak Technologies. Podle Jamese Goslinga „Java byla jednou z nejlepších možností spolu se Silk“. Protože Java byla tak jedinečná, většina členů týmu preferovala Javu než jiná jména. Jméno Java si vybral James Gosling při šálku kávy poblíž své kanceláře. Protože Java je ostrov v Indonésii, kde byla vyrobena první káva, jenž je nazývaná Java coffee. V ten samý rok byly vydány Alfa i Beta verze Javy. Také americký týdeník Time magazine označil Javu jako jeden z nejlepších produktů roku 1995. JDK 1.0 byl vydán 23. ledna 1996. Po prvním vydání Javy bylo do jazyka přidáno mnoho dalších funkcí. Nyní se Java používá v aplikacích Windows, webových aplikacích, podnikových aplikacích, mobilních aplikacích, kartách atd. Každá nová verze přidává nové funkce v Javě.

Seznam všech verzí Javy a datum vydání:

1 JDK Alpha and Beta - 1995

2 JDK 1.0 - 23. leden 1996

3 JDK 1.1 - 19. únor 1997

4 J2SE 1.2 - 8. prosinec 1998

5 J2SE 1.3 - 8. květen 2000

6 J2SE 1.4 - 6. únor 2002

7 J2SE 5.0 - 30. září 2004

8 Java SE 6 - 11. prosinec 2006

9 Java SE 7 - 28. červenec 2011

10 Java SE 8 - 18. březen 2014

11 Java SE 9 – 21. září 2017

12 Java SE 10 – 20. březen 2018

13 Java SE 11 - září 2018

14 Java SE 12 - březen 2019

15 Java SE 13 - září 2019

16 Java SE 14 - březen 2020

17 Java SE 15 - září 2020

18 Java SE 16 - březen 2021

19 Java SE 17 - září 2021

20 Java SE 18 - bude vydána v březnu 2022

3.3 Platformy Java

3.3.1 Platforma Java SE (Standart Edition)

Java SE je obecně používaná a nejrozšířenější verze platformy Java. Zároveň je verze platformy vyvíjená od první verze javy společností Sun Microsystems. Ale když zavedli termín platforma Java, potřebovali, kvůli původní kolekci API, je odlišit od ostatních

verzí. Obsahuje virtuální stroj, API knihoven základních funkcí a API knihoven pro vytváření základních funkcí (AWT – nejstarší, Swing – novější a do verze Java 11 to byla i Java FX).

3.3.2 Platforma Java ME (Micro Edition)

Jedná se o podmnožinu Java platformy SE, s cílem nabídnout sbírku API určenou pro vývoj software pro malá zařízení a zařízení s omezenými prostředky => Používá se v mobilních zařízeních. Java ME je rozdělena podle náročnosti API do dvou základních konfigurací. Každá konfigurace je navržena tak, aby představovala základní platformu pro dané zařízení, a není povoleno ji dále rozšiřovat. Tito konfigurace jsou CLDC a CDC.

3.3.3 Platforma Java EE (Micro Edition)

Java EE je určená pro vývoj a provoz podnikových aplikací a informačních systémů. Součásti platformy jsou definovány nad platformou Java SE. Součástí platformy jsou především specifikace pro vývoj webových aplikací - Java Servlet, Java Server Pages (JSP), JavaServer Faces (JSF), Contexts and Dependency Injection - vkládání závislostí, přístup k relačním databázím - Java Persistence API, vývoj sdílené business logiky - Enterprise Java Beans (EJB), přístup k legacy systémům - Java Connector Architecture (JCA), přístup ke zprávovému middleware - Java Messaging Services (JMS), komponenty zajišťující integraci webových aplikací a portálů - Portlety, podpora technologií Webových služeb.

3.3.4 Platforma Java Card

Java Card je cílena především do malých vestavěných zařízení a umožňuje uživateli programovat konkrétní aplikace pro konkrétní zařízení. Také je nezávislá na platformě a je kompatibilní se všemi standardy paměťových karet. Technologie Java Card byla původně vyvinuta pro účely zabezpečení citlivých informací uložených na čipových kartách. Zapouzdření dat, Applet firewall a šifrování jsou hlavní aspekty, které určují bezpečnost zařízení. Používá se převážně v kartách ATM a SIM.

3.4 Součásti platformy

3.4.1 JDK (Java Development kit)

je produktem Oracle Corporation, který obsahuje soubor základních nástrojů pro vývoj aplikací pro platformu Java. Někdy bývá označován jako Java SDK. Každý JDK vždy

obsahuje kompatibilní JRE a kompilátor (compiler).

3.4.2 JRE (Java Runtime Enviroment)

JRE nebo Java RTE, je vyvinut společností Sun Microsystems a zahrnuje prostředí JVM, knihovny kódů (API) a komponenty, které jsou nezbytné pro spuštění programů napsaných v prostředí Java. JRE je k dispozici pro více počítačových platforem, včetně Mac, Windows a Unix.

3.4.3 JVM (Java Virtual Machine)

JVM je sada počítačových programů a datových struktur, využívající modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java. Úkolem tohoto modulu je zpracovat pouze mezikód (Java bytecode). Java Byte Code je přeložen ze zdrojového kódu Javy pomocí kompilátoru, JIT (just-in-time) kompilátor pak převádí Java Byte Code do strojového kódu daného počítače. Pro spuštění v JVM musí být aplikace zkompilována do standardizovaného a přenosného binárního formátu, který je obvykle ve formě .class souborů, aplikace můžou ale být složeny z více různých souborů, proto je možné zabalit více javovských tříd do jednoho souboru typu .jar.

3.4.4 API (Application Programming Interface)

API je sbírka procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat. Určuje, jakým způsobem jsou funkce knihovny volány ze zdrojového kódu programu.

3.4.5 Knihovny

Knihovny jsou sada funkcí a procedur v podobě strojového kódu, které jsou uloženy do společného souboru. Tyto sady poté můžeme volat, abychom si jako programátoři usnadnili práci.

3.5 Vlastnosti Javy

Také známé jako Java buzzwords. Vytvoření těchto funkcí bylo učinit Javu přenosným, jednoduchým a bezpečným programovacím jazykem. Některé z nich hrají i důležitou roli v popularitě Javy. Mezi tyto funkce patří například:

Simple (jednoduchý) - Java se velmi snadno učí a její syntaxe je jednoduchá, čistá a snadno pochopitelná.

Object-oriented (objektově orientovaný) - Java je objektově orientovaný programovací jazyk a všechno v Jave je objekt. Objektově orientovaný znamená, že organizujeme náš software jako kombinaci různých typů objektů, které zahrnují jak data, tak chování.

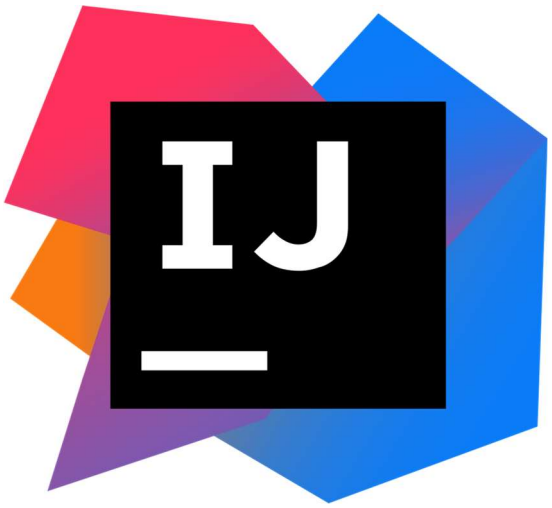
Platform Independent (nezávislý na platformě) - Java není závislá na určité platformě na rozdíl od C nebo C++. Platforma je hardwarové nebo softwarové prostředí, ve kterém běží program a dělí se na hardwarové a softwarové. Přičemž Platforma Java je softwarová, která běží nad ostatními hardwarovými platformami. Skládá se ze dvou složek Runtime Enviroment a API (Application Programming Interface).
Secured (zajištěný) - Java je známá především svou bezpečností. S Javou můžeme vyvíjet systémy bez virů. Java je zabezpečená, díky žádnému explicitnímu ukazateli, Java programy běží uvnitř karantény virtuálního stroje, Classloaderu, Verifikátoru bajtového kódu a Bezpečnostnímu manažerovi.
Robust (robustní) - význam slova robust je silný. Java je robustní, protože používá silnou správu paměti, chybí ukazatele, které se vyhýbají bezpečnostním problémům, Java poskytuje automatické shromažďování odpadků, které běží na JVM, aby se zbavilo objektů, které již Java aplikace nepoužívá, v Javě existuje zpracování výjimek a mechanismus kontroly typu.
Architecture-neutral (architektonicky neutrální) - Java je architektonicky neutrální, protože neexistují žádné funkce závislé na implementaci, například velikost primitivních typů je pevná.
Portable (přenosný) - Java je přenosná, protože vám usnadňuje přenos bajtového kódu Java na jakoukoli platformu. Nevyžaduje žádnou implementaci.
High-performance (vysoce výkonný) - Java je rychlejší než jiné tradiční interpretované programovací jazyky, protože bytecode Java je „blízko“ nativnímu kódu. Je stále o něco pomalejší než kompilovaný jazyk, jako například C++.
Distributed (Distribuovaný) - Java je distribuována, protože uživatelům usnadňuje vytváření distribuovaných aplikací v Javě.
Multi-threaded (více vláknové) - vlákno je jako samostatný program, který se spouští souběžně. Můžeme psát Java programy, které se zabývají mnoha úkoly najednou definováním více vláken. Hlavní výhodou multi-threadingu je, že nezabírá paměť pro každé vlákno. Sdílí společnou paměťovou oblast.
Dynamic (dynamický) - Java je dynamický jazyk. Podporuje dynamické načítání tříd. To znamená, že třídy jsou načteny na vyžádání. Podporuje také funkce ze svých rodných jazyků, tedy C a C++. Java podporuje dynamickou kompilaci a automatickou správu paměti (garbage collection).

4 Charakterizace vývojového prostředí (Ondřej Pták)

4.1 Úvod do IntelliJ

Je hned několik vývojových prostředí (pro Java placené a jejich neplacené verze nebo zcela zadarmo. Já stejně jako můj kolega jsme začínali ve vývojovém prostředí BlueJ, máme s ním několik zkušeností, a tak vývoj započal v něm, avšak brzy jsme zjistili, že BlueJ je příliš limitovaný pro náš program. Obsahoval příliš mnoho metod a kvůli tomu byl nestabilní („crashoval“), zasekával se a zkrátka nebyl zcela nejlepší volbou pro náš vývoj. BlueJ se spíš pro „začátečnické a amatérské“ práce. Těmito podmínkami jsme byli nuceni vybrat nové vývojové prostředí.

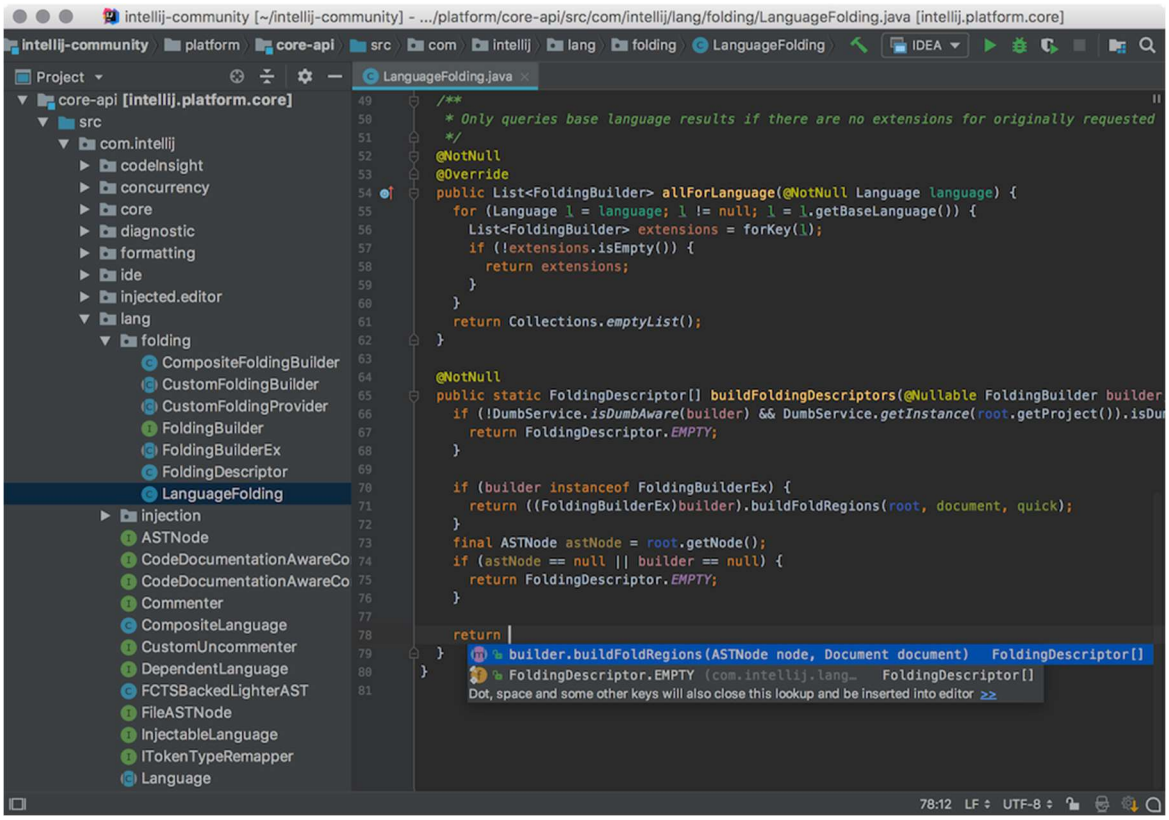
Na výběr nám zbyli vývojová prostředí jakožto: NetBeans, Eclipse, IntelliJ IDEA, Oracle JDeveloper, JUnit, PMD. Bohužel ani já tak můj kolega nemáme zkušenosti s těmito programy kromě IntelliJ, s tímto programe jsme byli seznámeni hned na začátku naše třetího ročníku na naší škole během hodin VAP (Vývoj Aplikací). Naskytla se příležitost si toto vývojové prostředí vyzkoušet, což značně ovlivnilo naší volbu. Sice se ještě nějaké funkce učíme za pochodu, náš program funguje plynule a stabilně a samotný vývoj je také snadnější.



Obrázek 1: obrázek loga IntelliJ

4.2 Šikovné vlastnosti IntelliJ

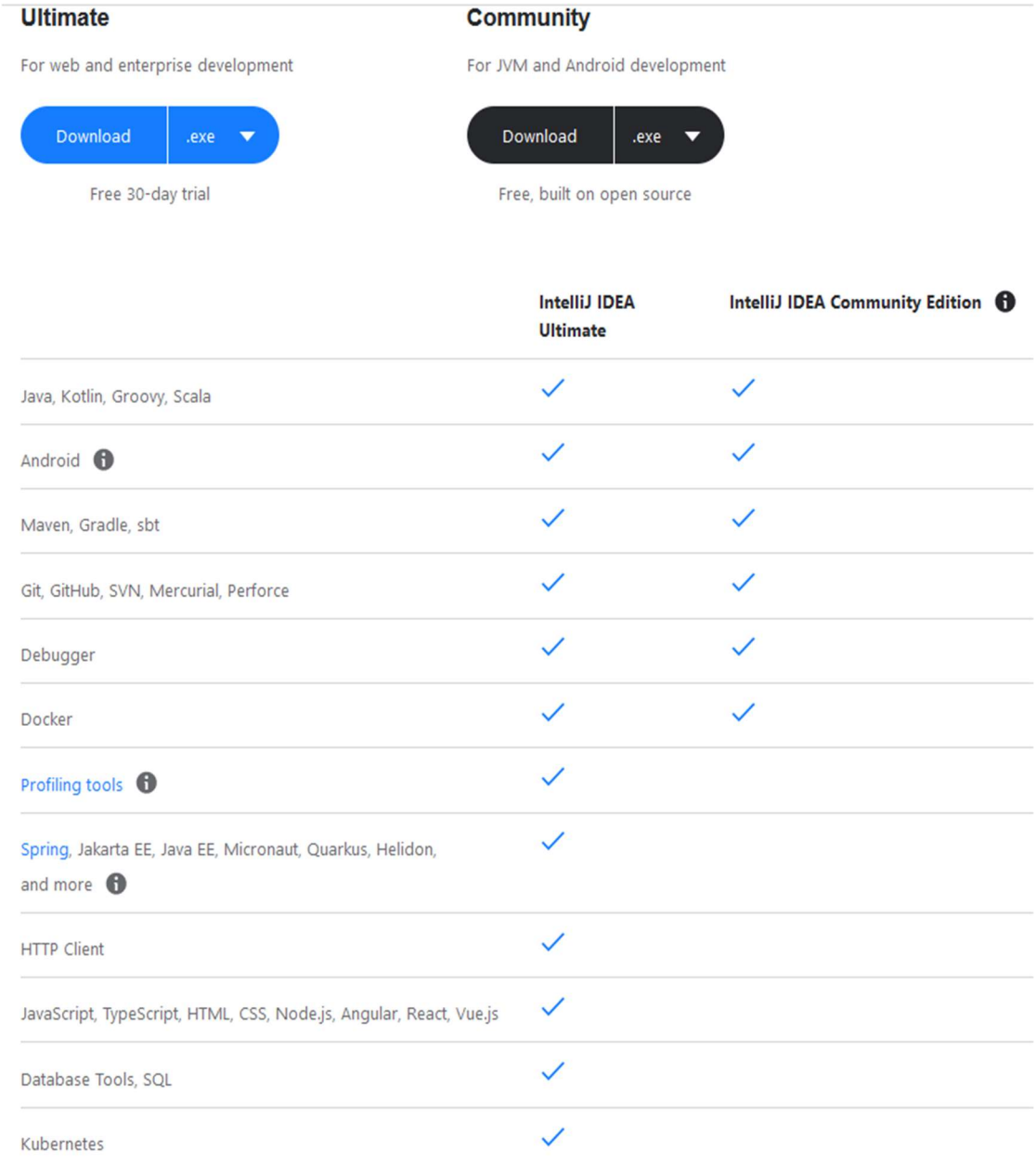
IntelliJ je vývojové prostředí určené především pro programovací jazyk Java. Také obsahuje pomocníka, který oskenuje váš source kód a snaží se programátorovi pomoc s relevantními nápady, autokorekcí kódu a našeptávačem. Snaží se napovědět programátorovi, doplňovat ho, avšak nijak nerušit a nepřerušovat průběh vývoje viz. obrázek č. 2. Hlavně z tohoto důvodu jsme se ho rozhodli vybrat, našeptávač se snaží porozumět našemu kódu a pomoct nám. Zjednodušuje zbytečně složité úseky kódu, také formátuje celou práci. Obsahuje hned několik klávesových zkratek například: Ctrl+Alt+L(formátování kódu); Ctrl+/(přidání/odstranění řádka s komentářem) a mnoho dalších.



Obrázek 2: snímek obrazovky (screenshot) vývoje programu ve vývojovém prostředí IntelliJ

Naši dosavadní práci jsme museli přepsat do IntelliJ, tento proces nám zabral nějaký čas, ale myslíme si, že toto rozhodnutí za to stálo. Nyní pokračujeme jedině ve vývojovém prostředí IntelliJ, přičemž používáme „Community“ verzi, která je neplacená a open-source. Na rozdíl od „Ultimate“ verze ji chybí framework Spring, HTTPs , JavaScript, CSS a další funkce především pro vývoj webových stránek. Tyto funkce jsou pro nás naštěstí zbytečné a nevyužili bychom je, takže k naší práci nám

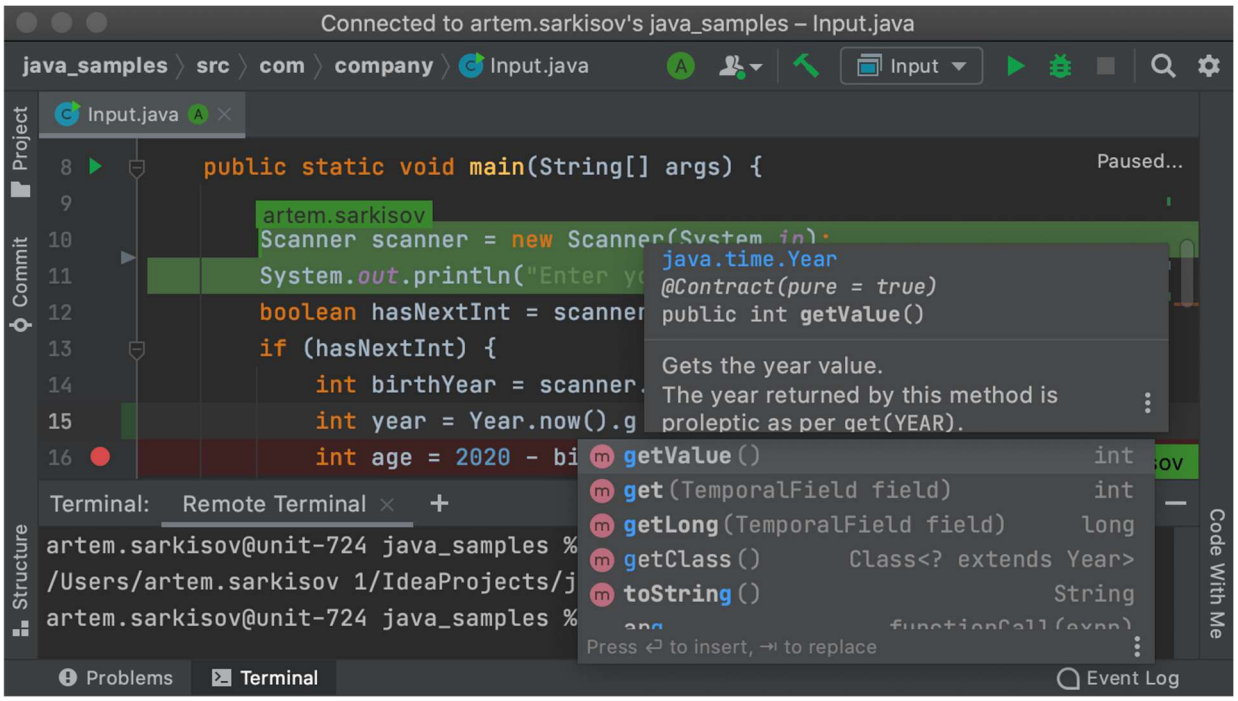
stačila neplacená verze programu viz. obrázek č. 3. Další velice nápomocná funkce IntelliJ je FXML editor, který ohromně usnadňuje a zrychluje vytváření grafického uživatelského prostředí programu („UI“).
viz. 4. bod „*Navrhnout vhodné grafické uživatelské prostředí editoru“*.



Obrázek 3: Porovnání IntelliJ IDEA Ultimate s Community edition

A rozhodně největší výhodu, kterou IntelliJ obsahuje je funkce ‚Code with me‘, díky které může společně pracovat na jednom projektu najednou viz. obrázek č. 4. Tato funkce nám nesmírně pomohla při vývoji a je to další odůvodnění proč jsme si vybrali IntelliJ a také nám pomohla zmírnit potíže, které nastaly, když náš GitLab přestal být

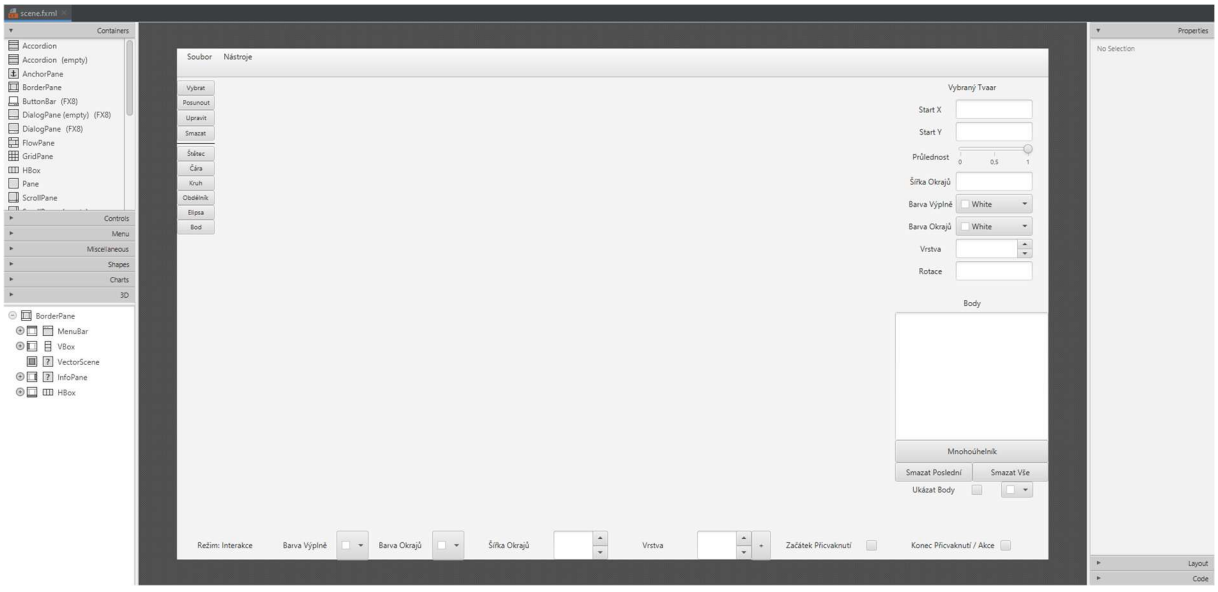
funkční. Stačilo aby můj kolega měl u sebe projekt otevřený a pozval mě přes funkci Code with me. Takže i když jsme nemohli vytvářet vlastni ‚commity' a ‚sprinty' přes GitLab, mohli jsme ho aspoň částečně nahradit. Tato funkce byla největším pilířem naší spolupráce, bez ní bychom nemohli náš program takto rychle i včas vytvořit.



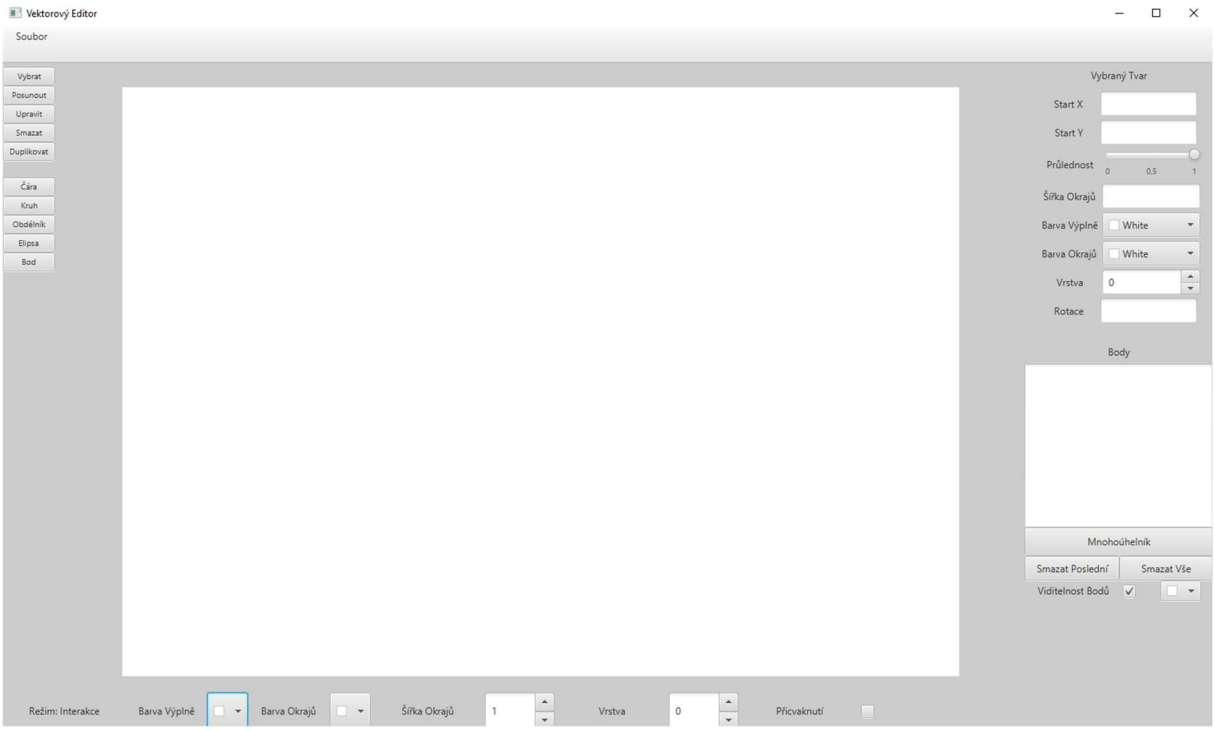
Obrázek 4: Funkce Code with me

5 Návrh grafického uživatelského prostředí

K vývoji programu jsme použili knihovnu Java FX. Java FX se v Javě používá jako GUI (Grafické uživatelské rozhraní) ‘framework‘, který má nahradit swing. Má totiž více funkcí a možností, jak ho využít na zobrazování komponenty, je neustále pod vývojem a je rozšiřováno o nové funkce. Java FX má své vlastní knihovny a v nich komponenty. Důležité komponenty Javy FX jsou například komponent ‘Stage‘, který funguje jako kontejner pro všechny objekty, komponent ‘Scene‘, na ten se přidává všechen použitý kontent a komponent ‘Layout‘ je kontejner, který nám skládá komponenty, uspořádává je a následně je vkládá na scénu. Pomocí Java FX můžeme vytvářet uživatelské rozhraní (dále již jen jako UI), 2D a 3D objekty, transformovat komponenty, efekty nebo i grafy. V našem případě jsme využili jeho vlastností na vykreslování určitých 2D objektů a jejich další úpravy. Z těchto důvodů jsme se rozhodli použít Javu FX pro vývoj našeho vektorového editoru, také protože Java Fx ‘framework‘ je podporován vývojovým prostředím IntelliJ a obsahuje ‘Scene builder‘, pomocí kterého lze velice rychle a efektivně vytvořit UI programu, bohužel často se objevuje problém, při kterém doplněk ‘Scene builder‘ nelze použít nebo nebo je nestabilní a zhroutí se, avšak tyto “neduhy“ výrazně nezpomalili náš vývoj editoru.



Obrázek 5: Příklad vývoje UI v ‘Scene builderu‘



Obrázek 6: finální verze UI vektorového editoru

6 Další použité technologie

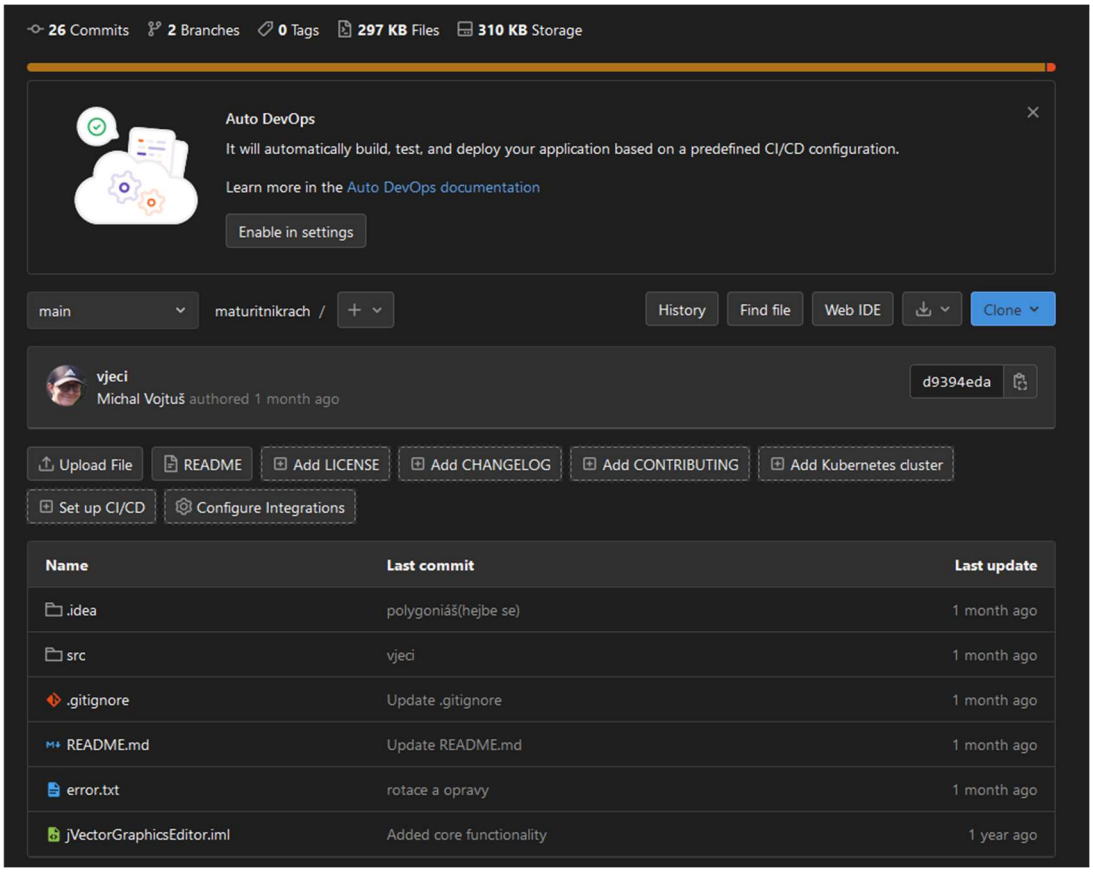
6.1 Školní Gitlab

GitLab je platforma DevOps, která organizacím umožňuje maximalizovat celkovou návratnost vývoje softwaru tím, že dodává software rychleji a efektivněji a zároveň posiluje zabezpečení a dodržování předpisů. S GitLabem může každý tým společně plánovat, budovat, zabezpečovat a nasazovat software pro rychlejší dosahování obchodních výsledků s naprostou transparentností, konzistencí a sledovatelností. GitLab je společnost s otevřeným jádrem, která vyvíjí software pro životní cyklus vývoje softwaru s odhadovaným počtem 30 milionů registrovaných uživatelů a více než 1 milionem aktivních uživatelů licencí a má aktivní komunitu více než 2 500 přispěvatelů. GitLab otevřeně sdílí více informací než většina společností a je standardně veřejný, což znamená, že o jejich projektech, strategii, směru a metrikách se otevřeně diskutuje a lze je nalézt na oficiálním webu. Týmová příručka GitLabu, pokud by byla vytištěna, měla více než 8 000 stran textu, je centrálním úložištěm toho, jak fungujeme, a je základním článkem hodnot GitLabu. Posláním GitLabu je udělat to tak, aby každý mohl přispět. Když může přispět každý, uživatelé se stanou přispěvateli a výrazně zvýšíme míru inovací.

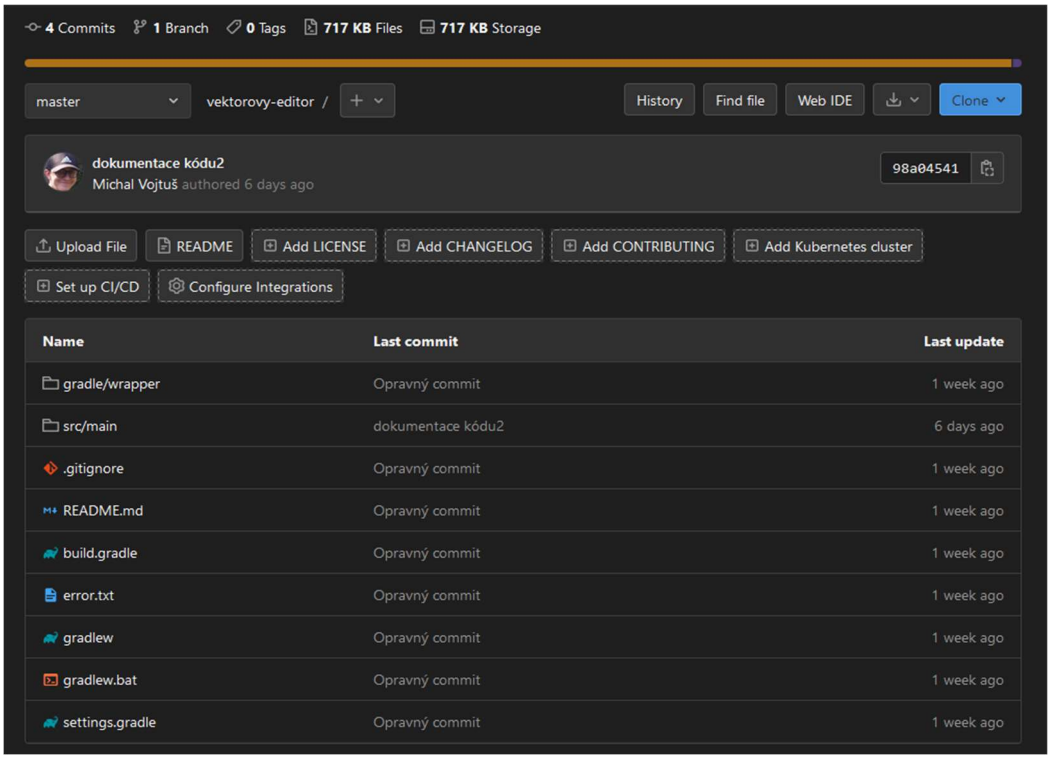
6.2 Naše zkušenosti s Gitlabem

Hned ze začátku jsme si vytvořili nás vlastní GitLab projekt, který jsme chtěli během našeho vývoje naše vektorového editoru používat, abychom mohli naší aplikaci verzovat, tím je myšleno, že náš vývoj programu je rozdělen do samostatných verzí. Díky těmto verzím lze předejít hned několika častým problémům při vývoji softwaru například: při nejnovějším commitu (= verzování aplikace, při každém commitu je nová verze aplikaci uložena na úložiště GitLabu a stará verze přepsána) nastal problém se softwarem a je z nějakého důvodu nefunkční, chybu se nedaří nalézt či ji odstranit. Může být třeba v kostře programu a k jejímu odstranění by bylo zapotřebí velké množství práce, či dokonce změnit fundamentální práci aplikace. Víme, že tyto problémy se mohou těžce odrazit na rychlosti vývoji aplikace, z tohoto důvodu byl GitLab jasnou volbou. Bohužel my jsme tyto velice praktické vlastnosti GitLabu nemohli využít naplno, protože během podzimu náš GitLab projekt přestal fungovat. Nešlo nám na něho posílat nové verze ani stahovat ty starší, toto byla velká rána pro náš vývoj, neboť jsme na GitLab spoléhali. Oba jsme měli na svých školních a vlastních

domácích zařízení měli stažené jiné verze naší práce, které jsme měli v plánu jednoduše jedním rychlým stažením nejnovější verze programu sjednotit nás vývoj. Takže jsme místo toho museli pracně stahovat na Flash disky a kopírovat z našich školních disků přes Filtr naší školy. Nakonec se nám to podařilo a stálo nás to drahocenný čas a na chvíli zastavilo náš vývoj softwaru. I přes tuto překážku jsme ihned pokračovali, abychom neztráceli čas. Poté jsme využívali funkci IntelliJ „Code with me“ jakožto menší náhradu. Projekt byl uložený na počítači Michala Vojtuše, který si ho pak skrze školní Filtr stahoval na svůj osobní počítač. Na oba počítače jsem se skrze již řečenou funkci připojoval a společně pracovali na našem vektorovém editoru. Na konci zimy Michala Vojtuše napadlo vytvořit nový GitLab projekt a zkusit to s ním. Nový projekt jsme vytvořili a doufali, že náš minulý problém opět nenastane a naštěstí opravdu nenastal. Na náš nově vytvořený náhradní projekt jsme ihned nahráli náš vektorový editor a mohli opět využívat benefitů GitLabu. Nakonec se nám podařilo zprovoznit i nás originální projekt. Od té doby již ho regulérně používáme, ale byl zprovozněn až moc pozdě aby hrál v našem vývoji jakkoliv signifikantní roli, protože hlavní část editoru již byla vytvořena. Posoudili jsme, že nás tento problém sice zpomalil, ale dokázali jsme pomoci a aspoň částečně ho vyřešit a měli jsme to štěstí, že se tento problém vyskytl takto brzo a ne když byla aplikace plně rozpracována.



Obrázek 7: Náhradní GitLab projek



Obrázek 8:Znovu zprovozněný GitLab projekt

7 Realizování jednotlivých funkcí grafického editoru (Ondřej Pták)

7.1 Kreslení základních tvarů

7.1.1 Úvod do problematiky

Mým úkolem bylo, aby náš program byl schopný vykreslit základní tvary jakožto čtyřúhelník, kruh elipsa atd., jak již bylo zmíněno v zadání. Všechny potřebné tvary jsou již předdefinovány v samotné Javě (čtyřúhelník/čtverec, trojúhelník, elipsa, kruh) a jsou vytvářeny pomocí metody draw(), která je doplněna o název kresleného obrazce např. drawOval() a parametry tvaru. Toto je základní stavební kámen našeho programu. Parametry tvaru je myšleno jeho souřadnice a jeho výška a šířka, u kruhu je to jeho průměr a u elipsy jsou tyto průměry dva. Protože chceme aby se náš tvar byl vytvořen přesně ve velikosti a tam kde uživatel chce aby byl, jeho parametry nesmí být předdefinovány. Nejdříve jsme vše vyvíjeli pomocí Java Swing a Graphics2D, jenže jsme brzy narazily no to, jak je Graphics2D omezené a spíše pro jednoduché kreslení např. Samotného obrázku složeného, které jsou předem vytvořeny programátorem na určitých souřadnicích. Jawa Swing má také omezené UI komponenty a je to opravdu stavěno spíše pro malé a jednoduché projekty, takže k naší práci to bylo zcela nevhodné. Proto jsme Swing opustili a začali náš projekt realizovat v JavaFX. JavaFX je nejnovější nástroj Javy pro vytváření UI a zároveň pro kreslení, což bylo právě to co jsme potřebovali, obzvlášť ten fakt toho, že IntelliJ je vývojové prostředí určeno pro vývoj UI viz 4. bod zadání. Poté co byl program pře-importován do IntelliJ, mohl náš vývoj nabýt finální podoby. Nejtěžší bylo vytvoření konceptu jak náš program bude fungovat a jak se bude samotné objekty vytvářet a editovat. Nakonec nás napadlo, že použijeme již vytvořené metody, avšak je bude rozšiřovat pomocí příkazu ‚extend'. To nám umožní zdědit všechny vlastnosti metod draw-objekt a tu změnit podle našich potřeb, například přidáním nových parametrů pro náš tvar. Díky tomuto jsme mohli najednou vytvářet všechny metody pro všechny objekty, protože se shodovali základní stavbou a často byli více či méně stejné nebo aspoň podobné. Všechny rozšiřovali jejich rodičovskou metodu draw a zároveň měli spoustu stejných metod pro editaci a interakci.

7.1.2 Samotný program

Jak již bylo řečeno v minulém pod-bodu, třídy, které se starají a tvorbu tvarů, se často shodují nejen v jejich samotné ‚kostře', ale i používají ty samé metody, které opět

rozšiřují a přepisují jak potřebují, protože není stejný postup při vyváření všechny tvarů, mají různé parametry, které potřebují. Hlavním takto odlišným tvarem je kruh a elipsa, neboť oba tvary potřebují a vlastní poloměr (elipsa potřebuje dva). Takže metody getCenterX() a setCenterX() jsou sice stejné v konceptu, ale prakticky jinak zapsané a lehce odlišné u každého tvaru. Tyto všechny metody jsou opět zapsány v jejich respektivním Java souboru nejen pro přehlednost, ale i pro bezpečnost a zabránění řetězové reakci při chybě v programu. Více o editačních metodách v 5. bodě zadání podbod d. Každý útvar má také svůj vlastní .java soubor přes který se vytváří a dostává parametry atd.viz. Obrázek našeho kódu č. 6.

Nyní k samotnému principu, je jednoduchý - uživatel stiskne a potáhne myší (drag and drop), podle jeho pohybu myší (do jaké výšky/šířky se posunul) se vytvoří daný útvar. Poté co uživatel klikne mód vytvoření tvaru se editor přepne do vytvářecího režimu a začne kreslit jeho požadovaný útvar, pomocí daných metod.

```
package custom_shapes;

import controller_components.ControllerScene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;

public class ShapeCircle extends Circle implements IShape {
    private ControllerScene scene;
    private int layer;
/*
*   Inicializace samotného objektu, a získávání všech nutných parametrů např. jeho souřadnice, barva a u kruhu a poloměr
*
* */
    public ShapeCircle(ControllerScene sc, int l, double ox, double oy, double r, Color stroke, Color fill, int strokeWidth) {
        scene = sc;
        layer = l;
        setCenterX(ox);
        setCenterY(oy);
        setRadius(r);
        setStroke(stroke);
        setFill(fill);
        setStrokeWidth(strokeWidth);
    }
```

Obrázek 9: Metoda pro vytváří kruhu

7.1.3 Popis funkce kódu

Zde na screenshotu kódu můžeme vidět hlavní metodu pro vytváření kruhu, tato metoda je veřejná, takže je všem přístupná, kruh je nejdříve přidán na danou scénu, poté si zjistí na jaké vrstvě má být. Ze základu je na vrstvě 0 a bude překrývat všechny starší tvary, tak jak se to normálně stává v Javě. Pokud by chtěl uživatel jeho vrstvu změnit,

jednoduše do pole vrstva napíše na jaké vrstvě má objekt být viz. obrázek č. 7. Dále metody setCenter, tyto metody získávají prvotní souřadnice objektu, respektive jeho začátek. Proměnné se jmenují ox a oy, což znamená originalX a originalY. Přesně tato metoda je stejná nejen konceptem, ale i praktickým zpracováním u všech tvarů, neboť zcela všechny tvary musí v nějakém bodě začínat. Dalším významným rozměrovým a jedinečným parametrem pro kruh a elipsu je poloměr (elipsa má tyto poloměry dva). Tyto proměnné jsou poté používány pro pohyb s tvarem, jeho editaci. Poté co je objekt nakreslen ihned uložen do pole objektů, to je požíváno při klikání na objekty, abychom mohli zjistit, zda-li na místo, kam uživatel kliknul, je nějaký tvar a pokud ano, tak který.

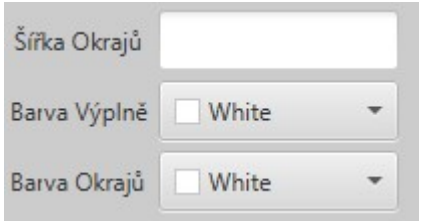
Obrázek 10: Volení vrstvy pro objekt

7.2 Změna barvy, typu a tloušťky čáry

7.2.1 Úvod do problematiky

Dalším možnost editace tvaru a zároveň jeho dalším parametrem, jak již bylo zmíněno, barva čáry neboli ohraničení. U tohoto bodu zadání se ovšem naštěstí žádné velké problémy nevyskytli, neboť řešení tohoto problému je triviální a značně rychlé. Každý tvar má ze základu barvu ohraničení černou a tloušťku 1, avšak tyto hodnoty se dají jednoduše změnit podle přání uživatele, stačí jen abychom získali uživatelovu preferovanou barvu a zvolenou tloušťku čáry. O tento je postaráno v UI sekci programu viz obrázky 8 a 9.

Obrázek 11: Výběr barvy výplně a okrajů při vytváření tvarů společně s šířkou okrajů



Obrázek 12: Výběř barvy výplně a okrajů, společně s šířkou okrajů při úpravě tvaru

7.2.2 Samotný kód a jeho popis funkce

Jak můžeme vidět na předchozích obrázcích, uživatel si jednoduše vybere z již v Javě vytvořeného ‘color pickeru’ svoji hledanou barvu a následovně do pole ‘Šířka Okrajů’ vepíše, jak moc široké ohraničení má být. Jak samotném kódu je toto provedeno, je vyjádřeno na obrázku číslo 13.

```
setStroke(stroke);
setFill(fill);
setStrokeWidth(strokeWidth);
```

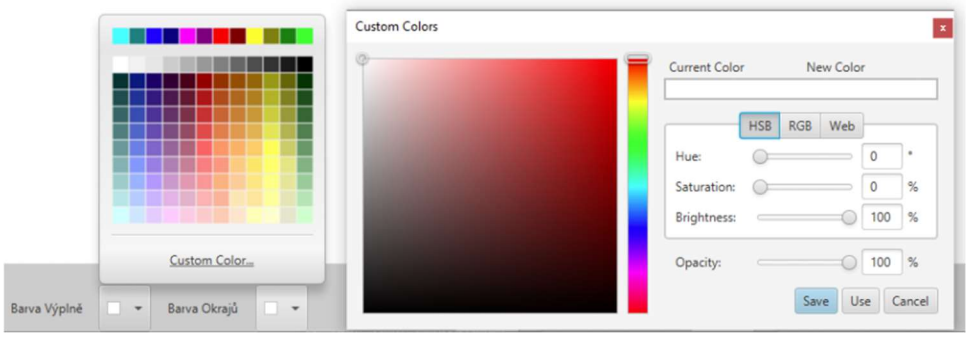
Obrázek 13: Detailní záběr na metody stroke a fill

Toto je opět další z několika parametrů našich tvarů. Jediná čára má toto jakožto hlavní parametr barvy, neboť výplň se nevztahuje na čáru. Barva čáry je přiřazena pomocí již vytvořené metody setStroke(), jejím parametrem je globální proměnná stroke, která získává svoji hodnotu z již zmíněného ’color pickeru’. Opět tato proměnná je ukládána do ’ArrayListu’ jakožto parametr každého tvaru, aby bylo možno ho později v režimu interakce změnit. Další metoda, která následuje, je setStrokeWidth(), která je také již před-definována. Jejím parametrem je globální proměnná strokeWidth, která je získávána od uživatele skrze UI komponentu. Na rozdíl od stroke je to celočíselná proměnná, čím větší číslo tím větší šířka okrajů.

7.3 Změna výplně tvaru

7.3.1 Úvod do problematiky

Změna výplně tvaru je samotným konceptem i svým zpracováním skoro identická jako změna barvy a tloušťky čáry. Jak již bylo řečeno, i u této části programu se nevyskytl žádný problém, neboť celý proces je opět řešen skrze před-definovanou metodou Fill(). Parametr této metody je barva, kterou bude daný tvar obarven. Ze základu je barva výplně bílá, ale lze ji změnit skrze její vlastní color picker. Postup ukládání a možné změny skrze mód interakce je velice podobný jako u editace čáry. Oba tyto parametry jsou stejně sdíleny mezi všemi tvary.



Obrázek 14: Míchání barev při tvorbě tvarů

7.3.2 Samotný kód a jeho popis funkce

Na předchozím obrázku číslo 11 je zobrazen detailní pohled na již několikrát zmíněný color picker, můžeme vidět, že kromě nějakých základních již vytvořených barev, může uživatel namíchat svoji vlastní. Dalším parametrem barvy je také opacita neboli její průhlednost. Tato vlastnost platí i pro barvu čáry, avšak je přístupná jen při míchání vlastní barvy. Průhlednost výplně a čáry se nemusí shodovat, avšak při měnění průhlednosti v režimu interakce je opacita měněna pro celý tvar naráz. Více o průhlednosti v kapitole g. Barvu pro výplně lze zadat nejen požitím RGB či HSB modelu, ale také i pomocí Web modelu např. #00ff00.

8 Realizování jednotlivých funkcí grafického editoru

Začal bych s metodou handle(), která je důležitá pro vykreslování jednotlivých tvarů, výběr tvarů pro další úpravy, nebo třeba pro pohyb tvarů.

8.1 Odstranění tvarů

První přidanou funkcí editoru, mimo kreslení samotných tvarů, bylo jejich mazání. Na toto původně nebyla potřeba metoda handle(), ale nyní díky ní program ví, že aktivovaný ClickMode (režim tlačítka myši) je DELETE. Základní odstraňování tvarů bylo přidáno na začátku vývoje vektorového editoru, kdy handle() ještě neexistovala. Samotná metoda removeShape() se zavolá po stisknutí tlačítka Smazat v UI editoru a kliknutí na tvar, který chceme odstranit. Metoda následně odstraní vrstvu společně s tvarem umístěným na dané vrstvě.

```
private void removeShape(IShape shape){
    for(ArrayList<IShape> layer : content){
        layer.remove(shape);
    }
    System.out.println(getChildren().remove(shape));
}
```

Obrázek 15: metoda removeShape pro odsranění tvaru

```
else if(action == ClickMode.DELETE){
  if(m.getPickResult().getIntersectedNode().getClass().getPackage().toString().equals(checkMovable)){
      IShape del = (IShape) m.getPickResult().getIntersectedNode();
      removeShape(del);
  }
```

Obrázek 16: část metody handle pro odstraňování tvarů

8.2 Změna velikosti a umístění

8.2.1 Změna umístění tvarů

Následně jsem přidal funkci pro pohybu tvarů, která využívá metody handle() stejně jako odstraňování tvarů. Takže po kliknutí na tlačítko Posun v UI editoru se ClickMode nastavý na MOVE a po kliknutí na tvar, který chceme posunout je proměnná isMoving nastavená na true. Po přesunutí tvaru myší a kliknutí na místo, kam chceme tvar přesunout se isMoving nastavý na false a pohyb je u konce.

Jako příklad zde mam metodu move() pro tvar čára. Čára se posouvá podle bodu, kterým ji nakreslíme, ten je přesunut kurzorem myši, a druhý bod přímky je vypočítán

podle původní vzdálenosti od začínajícího a konečného bodu.

```
@Override
public void move(double ox, double oy){
    double diffX = getStartX() - getEndX();
    double diffY = getStartY() - getEndY();
    setStartX(ox);
    setStartY(oy);
    setEndX(ox - diffX);
    setEndY(oy - diffY);
}
```

Obrázek 17: metoda move pro posun čáry

```
} else if(action == ClickMode.MOVE){
    if(m.getButton() == MouseButton.PRIMARY && !isMoving){
        if(m.getPickResult().getIntersectedNode().getClass().getPackage().toString().equals(checkMovable)){
            currentShape = (IShape) m.getPickResult().getIntersectedNode();
            infoPane.setShape(currentShape);
            xTmp = currentShape.getStartX();
            yTmp = currentShape.getStartY();
            System.out.println("začátek pohybu");
            isMoving = true;
        }
    } else if(m.getButton() == MouseButton.PRIMARY && isMoving){
        System.out.println("konec pohybu");
        isMoving = false;
    } else if(m.getButton() == MouseButton.SECONDARY && isMoving){
        stopMovingShape();
    }
```

Obrázek 18: Část metody handle pro pohyb tvarů

8.2.2 Změna velikosti tvarů

Změna velikosti patří mezi další funkce využívající metodu handle(). Poté, co klikneme na tlačítko Upravit v UI editoru je nastaven ClickMode na ADJUST, následně klikneme na tvar, který chceme upravit a poté začneme tvar znovu vykreslovat. A proměnná isAdjusting je nastavena na true. Tímto upravujeme pouze pozici určitých bodů, takže jejich atributy jako například barva, šířka okrajů zůstanou stejné. V případě čáry zůstane první bod a bod druhý je vytvořen na místě, kam klikneme. Pro kruh a elipsu zůstane střed a my nastavujeme nový poloměr. Při úpravě čtyřúhelníku je ponechán levý horní bod a my upravujeme pravý spodní bod. Po dokončení úpravy je isAdjusting nastavená na false.

```java
@Override
public void adjust(double mx, double my){
    if(mx < originX){
        setX(originX + (mx - originX));
        setWidth(Math.abs(getX() - originX));
    } else{
        setX(originX);
        setWidth(Math.abs(mx - originX));
    }
    if(my < originY){
        setY(originY + (my - originY));
        setHeight(Math.abs(getY() - originY));
    } else{
        setY(originY);
        setHeight(Math.abs(my - originY));
    }
    System.out.println("X: " + getX() + "\tY: " + getY() + "\nWidth: " + getWidth() + "\tHeight: " + getHeight() + "\n");
}
```

Obrázek 19: metoda adjust pro úpravu čtyřúhelníku

```java
public double getAdjustX(){
    if(originX > getX()){
        return originX - getWidth();
    } else{
        return originX + getWidth();
    }
}

public double getAdjustY(){
    if(originY > getY()){
        return originY - getHeight();
    } else{
        return originY + getHeight();
    }
}
```

Obrázek 20: getry pro souřadnice nového bodu čtyřúhelníku

```java
else if(action == ClickMode.ADJUST){
     if(m.getButton() == MouseButton.PRIMARY && !isAdjusting){
        if(m.getPickResult().getIntersectedNode().getClass().getPackage().toString().equals(checkMovable)){
            System.out.println("začátek úpravy");
            currentShape = (IShape) m.getPickResult().getIntersectedNode();
            infoPane.setShape(currentShape);
            xTmp = currentShape.getAdjustX();
            yTmp = currentShape.getAdjustY();
            isAdjusting = true;
        }
     } else if(m.getButton() == MouseButton.PRIMARY && isAdjusting){
        System.out.println("konec úpravy");
        isAdjusting = false;
     } else if(m.getButton() == MouseButton.SECONDARY && isAdjusting){
        stopAdjustingShape();
     }
```

Obrázek 21: část metody handle pro úpravu tvarů

8.3 Export do souboru

Poslední funkcí programu mělo být exportování výsledného obrázku do bitmapového souboru, tedy souboru s příponou png nebo jpg. Tuto část kódu jsem našel na ‘Stack Overflow‘ Prakticky je toto zadání řešeno skrze použití instance třídy File, přesněji použití ‘FileChooser‘. Na horní liště UI editoru se nachází menu Soubor**,** kde najdeme možnost pro uložení plátna s názvem Uložit**.** Po stisknutí na tlačítko Uložit se otevře ‘FileChooser‘ a nám stačí napsat jen jméno souboru s příponou, podle formátu do kterého chceme výtvor exportovat.

```
//metoda pro uložení do png
    @FXML
    public void save(){
        FileChooser savefile = new FileChooser();
        savefile.setTitle("Uložit");
        File file = savefile.showSaveDialog(Main.stage);

        if (file != null) {
            try {
                WritableImage writableImage = new WritableImage( width: 1024,  height: 720);
                scene.snapshot( params: null, writableImage);
                RenderedImage reimg = SwingFXUtils.fromFXImage(writableImage,  bimg: null);
                ImageIO.write(reimg,  formatName: "png", file);
            } catch (IOException ex) {
                System.out.println("Error!");
            }
        }
    }
```

Obrázek 22: Metoda save pro export souboru

8.4 Duplikace tvarů

Když uživatel zapne režim duplikace pomocí komponent v UI editoru a následným kliknutím na objekt, který chce duplikovat, jej vybere je zavolána klonovací metoda daného tvaru, neboť každý typ tvaru se liší parametry a musí mít unikátní metodu, která duplikaci zajistí.

Tato funkce zkopíruje všechny atributy tvaru, na který klikneme, a vytvoří na jeho místě nový identický tvar na stejné vrstvě, avšak kvůli hierarchii vykreslovaných objektů v Javě bude umístěn nad původním tvarem a překrývat všechny starší tvary a tvary na nižších vrstvách. Stejně jako předchozí funkce pracuje s metodou handle() a ClickMode se nastaví na režim DUPLICATE**.** A po kliknutí na tvar se vytvoří jeho klon jako nový tvar na plátně.

```
public ShapeCircle clone(){
    ShapeCircle c = new ShapeCircle(scene, layer, getCenterX(), getCenterY(), getRadius(), (Color) getStroke(), (Color) getFill(), (int) getStrokeWidth());
    c.setRotate(getRotate());
    c.setOpacity(getOpacity());
    return c;
}
```

Obrázek 23: Metoda clone pro kruh

```
else if(action == ClickMode.DUPLICATE){
  System.out.println("DUP");
  if(m.getPickResult().getIntersectedNode().getClass().getPackage().toString().equals(checkMovable)){
      IShape toDuplicate = (IShape) m.getPickResult().getIntersectedNode();
      newShape(toDuplicate.clone());
  }
```

Obrázek 24: Část metody handle pro duplikaci

9 Závěr

Nakonec bych chtěli shrnout nejen finální produkt našeho projektu, ale také i jeho průběh. Začali bychom průběhem, když na něj nyní podíváme s odstupem času, tak jsme zjistili, že nebyl plynulý. Největší “brzdou” pro nás byl špatné zvolení Frameworku swing. Import projektu do JavaFX nám sebralo nemálo času, avšak jsme stále názoru, že se toto rozhodnutí vyplatilo. Naopak největší posun jsme zaznamenali během listopadu a zimy, v tomto období jsme “dopilovali” všechny funkce programu a splnili zadání práce, dokonce jsme některé funkce přidali navíc, jakožto samotné vrstvení objektů, jejich rotace, průhlednost. Důvod vytvoření těchto funkcí nebylo z důvodu, že bychom měli nadbytek času, ale spíše že je bylo snadné integrovat již s nějakou funkcí. Průhlednost je spjatá s barvou, vrstvy jsou obecně spojené se samotnou tvorbou objektů a rotaci jsme přidali, protože jsme byli názoru, že rotace spadá pod bod “editace tvarů”. V budoucnu bychom rádi pokračovali na vývoji našeho editoru, změněním UI a celkově udělat editor více uživatelsky přívětivým, přidáním změny kurzoru při různých režimech kreslení, export plátna do svg souborů. Zkrátka obecně udělat rozhraní více profesionální.

10 Seznam literatury

Java (platforma). Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2022 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Java_(platforma)

History of Java. Javatpoint: Tutorials List [online]. Indie: issuu, 2021 [cit. 2022-04-18]. Dostupné z: https://www.javatpoint.com/history-of-java

Patrick Naughton. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2022 [cit. 2022-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Patrick_Naughton

Java Tutorial. Javatpoint: Tutorials List [online]. Indie: issuu, 2021 [cit. 2022-04-18]. Dostupné z: https://www.javatpoint.com/java-tutorial

Java Virtual Machine. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2022 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Java_Virtual_Machine

JDK. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2021 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/JDK

Java Card. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2021 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Java_Card

Java ME. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2021 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Java_ME

Java SE. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia Foundation, 2021 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Java_SE

Java EE. Wikipedia: the free encyclopedia [online]. Kalifornie: Wikimedia

Foundation, 2021 [cit. 2022-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Jakarta_EE

IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains [online]. Česká republika: JetBrains, 2022 [cit. 2022-04-19]. Dostupné z: https://www.jetbrains.com/idea/

About GitLab. GitLab: Iterate faster, innovate together [online]. GitLab, 2022 [cit. 2022-04-19]. Dostupné z: https://about.gitlab.com/company/

JavaFX: Save view of pane to image [duplicate]. Stack Overflow [online]. New York City: Stack Exchange, 2016 [cit. 2022-04-18]. Dostupné z: https://stackoverflow.com/questions/38028825/javafx-save-view-of-pane-to-image

11 Seznam obrázků