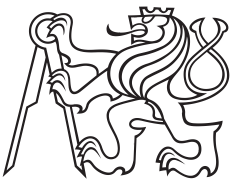
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Aplikace pro podporu hromadného odesílání korespondence

Vladimír Dvořák
Studijní program: Otevřená informatika, Obor: Softwarové systémy

Leden 2018

Vedoucí práce: doc. Ing David Šišlák Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Dvořák Vladimír

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: Aplikace pro podporu hromadného odesílání korespondence

Pokyny pro vypracování:

Navrhnete a implementujte aplikaci pro podporu hromadného odesíláni dopisu a balíku. Aplikace bude umožňovat spravovat seznam adres a vytvářet různé typy tiskových sestav (nalepovací statky, obálky, poštovní arch). Vzhled aplikace bude vytvořen takovým způsobem, aby umožnoval intuitivní použiti uživatelem znalým práce se základními programy.
1) Analyzujte existující programy pro takovýto typ úlohy.
2) Navrhnete funkcionalitu programu a zdokumentujete všechny způsoby použiti funkci pomoci UML diagramu s ohledem na výsledky analýzy z bodu 1.
3) Navrhnete vhodnou strukturu programu a tuto popište.
4) Implementujte aplikaci v jazyce JAVA. Aplikace bude obsahovat příslušné testy? Jednotkové i integrační.
5) Vyzkoušejte intuitivnost aplikace na uživatelích ve všech způsobech použiti programu. Tyto testy zdokumentujte a vyhodnoťte.
6) Navrhnete budoucí vylepšení aplikace.

Seznam odborné literatury:

[1] Schildt, H.: Mistrovstvi Java. Computer Press, 2014.
[2] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design patterns. Addison-Wesley, 1994.
[3] Fowler, M.: Destilovane UML. Grada, 2009.Smetana. 2016. Next generation of Second-Screen. Realtime application MyICPC.

Vedoucí: doc. Ing. David Šišlák, Ph.D.

Platnost zadání do konce zimního semestru 2018/2019

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 23.3.2017

Poděkování / Prohlášení

Chtěl bych poděkovat panu doc. Ing. Davidu Šišlákovi Ph.D. za velmi užitečné rady a konzultace, které mi pomohly při vypracování mé bakalářské práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 9. 1. 2018

.....

Abstrakt / Abstract

Tato práce se zabývá návrhem, implementací a testováním aplikace pro zapisování poštovních adres podle základních principů softwarového inženýrství. Hlavní téma je vytvoření takového programu, který bude uživateli usnadňovat práci při zapisování adres současně na různé druhy papírů, například na nalepovací štítky, obálky, a poštovní arch. Zjednodušuje doručování, zkracuje čas při psaní adresátů a tím pádem zrychluje práci jednotlivých pracovníků v celém procesu doručování.

Klíčová slova: PostManager, pošta, java, java GUI.

This thesis will focus on a proposal of an implementation and testing of application made for inserting addresses based on basic engineering principles. The main theme is to create a program which will allow an easier way of inserting data simultaniously on various types of paper, e.g. self-adhesive labels, evelopes and post sheet. It shortens the time of delivery and the time spent writing out addressee, therefore it speeds up the work done by everyone who is included in the whole delivery process.

Keywords: PostManager, post, java, java GUI.

Obsah /

Obrázky

Kapitola 1
Úvod

1.1 Motivace práce a cíle

Tato práce se bude zabývat návrhem, implementací a testováním aplikace pro zapisování poštovních adres podle základních principů softwarového inženýrství. Hlavní téma je vytvoření takového programu, který bude uživateli usnadňovat práci při zapisování adres na různé druhy papírů současně. Například na nalepovací štítky, obálky a poštovní arch.

K vytvoření této aplikace mě motivovala naše rodinná firma, která ročně odešle cca tisíc poštovních zásilek. Největší problém měla s vypisováním poštovních archů. Problematika spočívá v tom, že pokud chceme zákazníkovi odeslat doporučený dopis nebo balík do ruky, je potřeba na obálku vyplnit adresáta, odesílatele a zároveň vyplnit poštovní arch nebo podací lístek. Na trhu není žádná aplikace, která by toto dělala automaticky. Pro uživatele této aplikace by bylo výhodou zapsání adresáta na obálku a adresáta na poštovní arch pouze v jednom kroku. Zároveň by se mu mohl automaticky na obálku vepsat také odesílatel. Tím si ušetří celkem tři kroky a čas práce se zkrátí o dvě třetiny. Aplikace by mohla ušetřit podnikateli čas a jak víme tak i peníze. Aplikace by umožňovala vkládat vyplněného adresáta a odesílatele automaticky na určená místa na obálce. Stejný postup by existoval při vyplňování adres na nalepovací štítky. Uživatel si zvolí velikost a počet štítků na stránce. Pak by stačilo jen vyplňovat adresáty a ty by se uživateli umisťovaly na jednotlivé štítky na zvoleném archu.

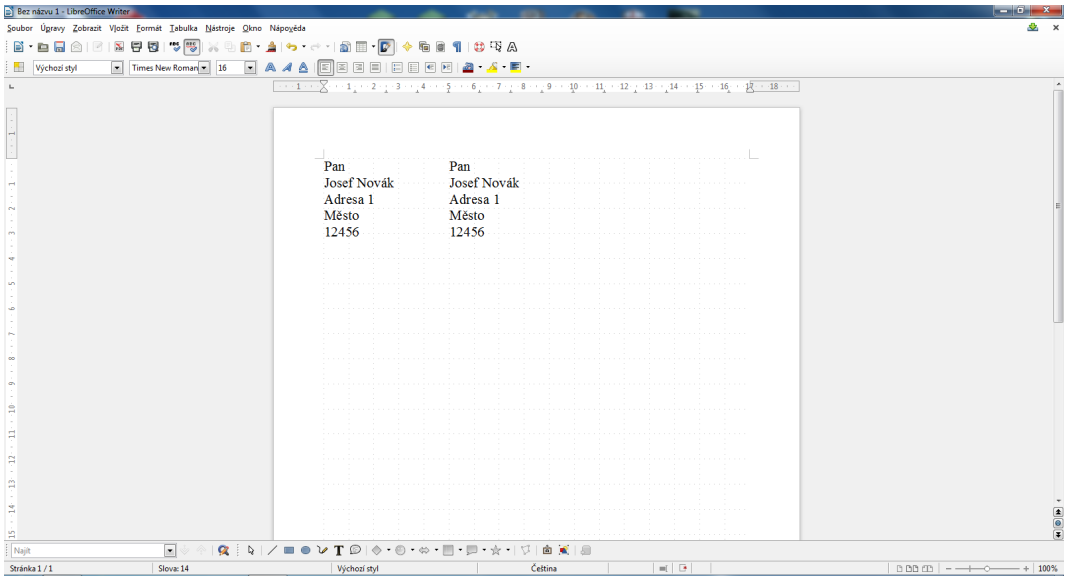
Tuto aplikaci mohou využít firmy a státní organizace, kteří korespondují se svými zákazníky a partnery Českou poštou ve velkém objemu. Příklad můžeme uvést na bytovém družstvu. Družstvo spravuje celkem 4 domy po 5 vchodech a v každém vchodu je 12 nájemných bytů a je potřeba je informovat o důležitém stanovisku. Družstvo tím pádem musí obeslat všech 240 členů. S touto aplikací by stačilo v prvním kroku vyplnit odesílatele a v druhém kroku vložit adresy členů družstva. Aplikace rozmístění adresátu a odesílatelů provede automaticky na obálky a poštovní archy.

Žádná z aplikací, která je dostupná na trhu tuto funkci neumožňuje. Kdybych to měl shrnout do jednoho bodu, v čem je tato aplikace lepší, tak je to případ, kdy uživatel zapíše jednu adresu a tato adresa se mu automaticky promítne na všechny zvolené druhy papírů. Je možno použít mnoho dostupných aplikací, ale vždy je potřeba vyplnit adresáta zvlášť na obálku a zvlášť na podací arch. Pro zjednodušení se může adresát kopírovat a vkládat, ale je to další zbytečná zdlouhavá činnost. Podobně mohou tuto aplikaci využívat e-shopy, které vyřizují zakázky převážně zásilkovou dopravou k zákazníkovi.

1.2 Existující programy

Informace jsou převzaty až na výjimky z jednotlivých stránek uvedených programů.

1.2.1 LibreOffice

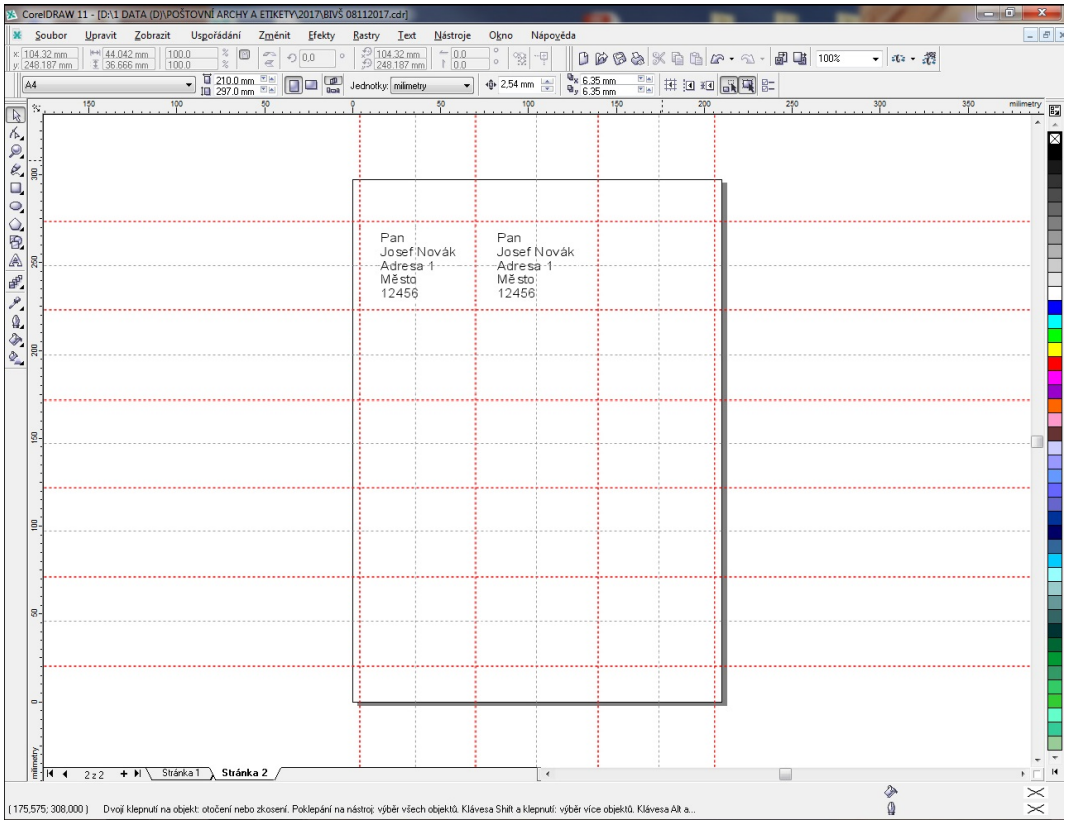


Obrázek 1.1. LibreOffice

Název: LibreOffice [1]
Výrobce: The Document foundation
Verze: 5.4.3
Výstupní formát: odt
Popis: LibreOffice 1.1 je kancelářský balík šířený jako svobodný software. Je k dispozici pro různé platformy v mnoha jazycích. Balík se skládá z textového procesoru, tabulkového procesoru, grafického editoru, prezentačního nástroje, databáze a nástroje pro zápis matematických výrazů. Jméno programu je spojeno ze slov libre - svobodný a office - kancelář. Je dostupný na Windows, MacOS, Linux, Solaris a FreeBSD. Pro zapsání adresy si uživatel celé formátování textu musí zvolit sám. Musí si sám odřádkovat jednotlivé řádky adresy, zvolit písmo, aby se vešlo na štítek, vždy musí ručně zvolit pozici adresy a nemá zde ani jistotu, že se mu adresa vytiskne přesně na štítek. V tomto programu nelze vytvořit šablonu, do které by se adresy samy pozicovaly. Při každém novém listě se musí adresy umístit na dané pozice ručně.

[1] `https://cs.libreoffice.org/`

1.2.2 CorelDRAW

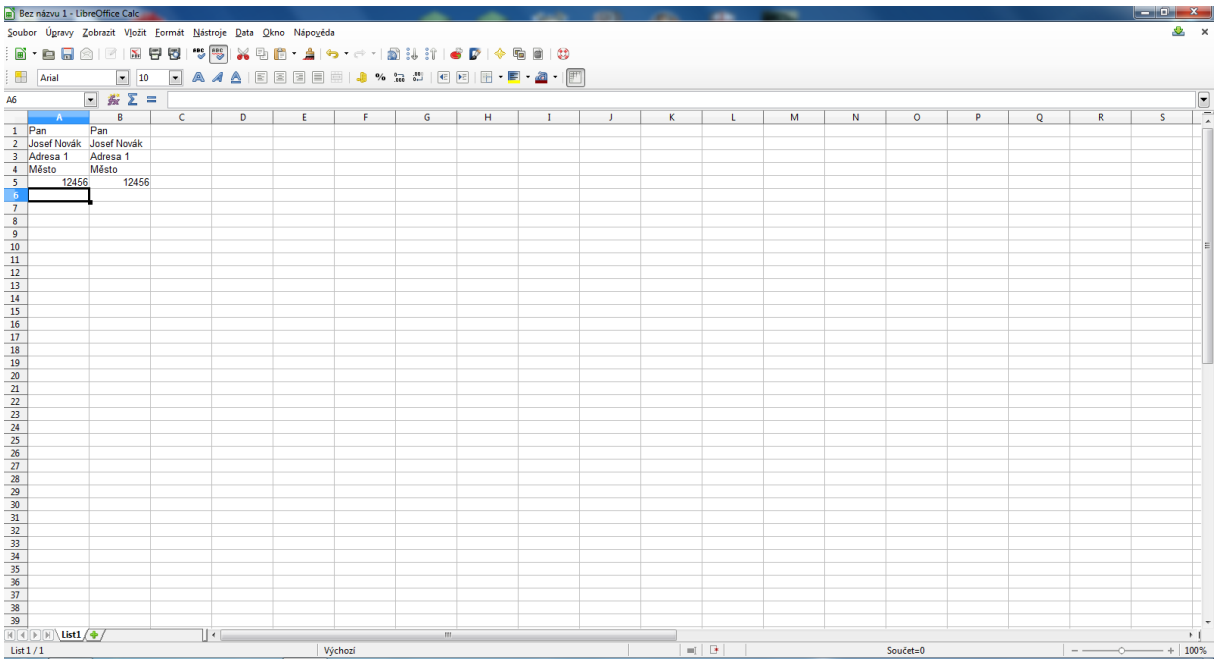


Obrázek 1.2. CorelDraw

Název: CorelDRAW [1]
Výrobce: Corel Corporation
Verze: X8
Výstupní formát: cdr
Popis: CorelDraw 1.2 je vektorový grafický editor. Je to aplikace pro vektorové ilustrace a stránkový zlom. Je dostupný na Windows. K dispozici je i český jazyk. Zde je postup úplně stejný, jako je u programu LibreOffice. Oproti programu LibreOffice jsou zde navíc pomocné linky, které uživateli ukazují, kam dané adresy psát, aby se mu správně vytiskly. Výhoda těchto linek je, že si je můžete uložit pro další použití. Nevýhoda těchto linek je na úplném počátku práce. Tyto linky si musí uživatel navolit ručně a musí vyzkoušet pár adres na papíru, jestli se mu srovnaly na štítky a pokud se mu to nepovede, příjde tak zbytečně o papír. V tomto programu také nelze vytvořit šablonu, do které by se adresy samy pozicovaly a při každém novém listě se musí adresy umístit na dané pozice ručně.

[1] `https://www.coreldraw.com/cz/`

1.2.3 Excel

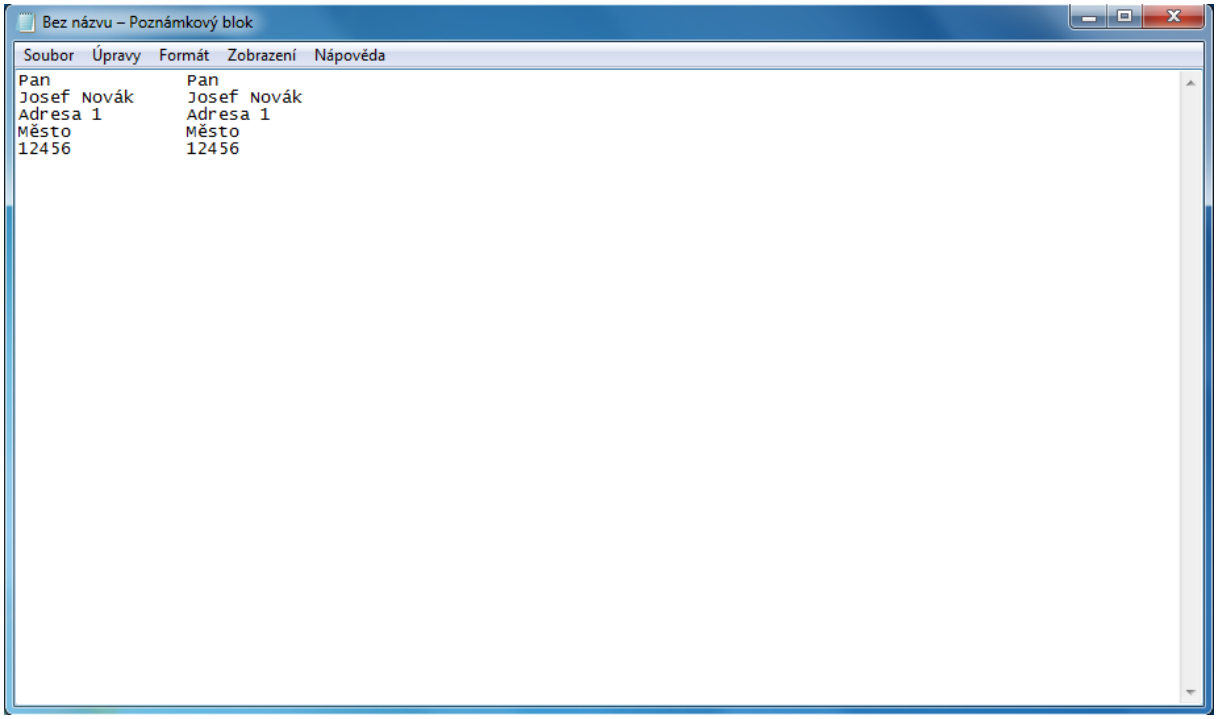


Obrázek 1.3. Excel

Název: Microsoft Excel [1]
Výrobce: Microsoft
Verze: 2016
Výstupní formát: ods
Popis: Microsoft Excel 1.3 je tabulkový procesor pro operační systém Windows a počítače Macintosh. Dnes je součástí kancelářského balíku Microsoft Office. Zde si uživatel může adresy zapsat přesně na řádky. Nevýhoda je , že zde hned nevidí, jakým způsobem tyto adresy budou rozloženy na papíru.

[1] `https://products.office.com/cs-cz/excel`

1.2.4 Notepad



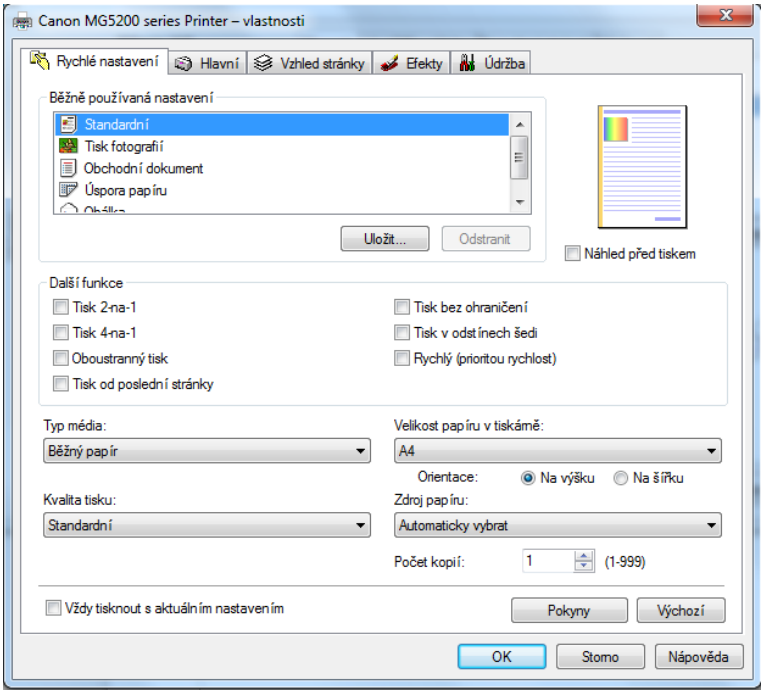
Obrázek 1.4. Notepad

Název: Notepad - Poznámkový blok [1]
Výrobce: Microsoft
Verze: 2016
Výstupní formát: txt
Popis: Poznámkový blok 1.4 je jednoduchý textový editor obsažený v Microsoft Windows. Protože editor neobsahuje formátovaný text, není tento program vhodný pro zápis adres. Adresy jsou v Poznámkovém bloku rozmístěny náhodně. Uživatel zde nemá jistotu správného tisku. Umístit adresy do správných pozic je zde velmi obtížné. Pro usnadnění práce se může projekt uložit jako nový projekt a adresy v něm přepsat.

1.2.5 Software instalovaný v ovladačích počítače

Zde bych pouze uvedl, že ovladače k tisku umí umístit a tisknout adresy na obálky, ale to neřeší problém současně zapsat adresu na poštovní arch nebo na štítky, to musí udělat uživatel zvlášť 1.5. A tento problém chci v bakalářské práci řešit.

[1] `https://cs.wikipedia.org/wiki/`



Obrázek 1.5. Tisk

1.3 Shrnutí

Ve všech výše uvedených programech lze samozřejmě dosáhnout požadovného cíle, ale za velmi dlouhou dobu. Cílem této bakalářské práce je vyvinout takový pogram, který minimalizuje ztrátu času při zápisu adres k rychlému vytisknutí.

1.4 Kapitoly práce

Práce je rozdělena do osmi kapitol. První kapitola se zabývá motivací a cílem této práce. Druhá kapitola obsahuje funkcionalitu programu. Třetí kapitola se věnuje návrhu celé aplikace a obsahuje UML diagramy, které popisují navržení celého systému. Čtvrtá kapitola se zabývá implementací a je doplněna o obrázky aplikace. Pátá kapitola se věnuje databázi. Šestá kapitola se zabývá testováním aplikace za použití JUnit testů a integračních testů. Sedmá kapitola obsahuje testování softwaru z pohledu uživatele. Osmá kapitola se věnuje závěrečnému slovu a budoucímu zlepšení aplikace.

Kapitola 2
Funkcionalita programu

2.1 Požadavky

V této kapitole se budu zabývat funkčními a nefunkčními požadavky celé aplikace. Požadavky na systém popisují, co má daný systém dělat, jaké služby uživateli poskytuje a jaká jsou omezení jeho činnosti. Tyto požadavky odrážejí potřeby zákazníka na systém, který bude sloužit k určitému systému. Text je převzatý z literatury [1].

2.2 Funkční požadavky

U funkčních požadavků se jedná o popis služeb, které by měl systém poskytovat, reakce systému na určité vstupy a chování systému v daných situacích.

2.2.1 Standardní funkcionalita

- **Otevřít aplikaci:** Systém bude umožňovat uživateli otevření aplikace.
- **Založit projekt:** Systém bude umožňovat uživateli založení nového projektu. Při založení projektu musí uživatel vybrat tyto možnosti.
  - Název
  - Druh papíru nebo obálky
  - Rozměry papíru nebo obálky
  - Odesílatele
  - QR kód pro odesílatele
- **Vybrat místo pro ukládání projektů:** Systém bude umožňovat uživateli vybrání místa, kam se mu budou ukládat projekty.
- **Načíst projekt:** Systém bude umožňovat uživateli načtení rozpracovaného projektu.
- **Uložit projekt:** Systém bude umožňovat uživateli uložení projektu.
- **Přeuložit projekt:** Systém bude umožňovat uživateli přeuložení projektu.
- **Tisknout papír:** Systém bude umožňovat uživateli tisk papírů.
- **Ukončit projekt:** Systém bude umožňovat uživateli zavření projektu.
- **Ukončit aplikaci:** Systém bude umožňovat uživateli ukončení aplikace

2.2.2 Adresa

- **Pracovat s adresou adresáta:** Systém bude umožňovat uživateli práci s adresou adresáta. Při práci s adresou musí uživatel vyplnit tyto údaje.
  - Jméno
  - Příjmení
  - Adresa
  - Město

  - Poštovní směrovací číslo

- **Pracovat s adresou firmy:** Systém bude možňovat práci s adresou firmy. Při práci s adresou firmy musí uživatel vyplnit tyto údaje.

  - Jméno
  - Adresa
  - Město
  - Poštovní směrovací číslo

- **Přidat odesílatele:** Systém bude umožňovat uživateli přidat odesílatele.
- **Kontrola adresy:** Systém bude umožňovat uživateli kontrolu správnosti adresy přes Open street map.
- **Generovat QR kód adresy:** Systém bude umožňovat uživateli generovat QR kód dané adresy.

2.2.3 Papír

- **Pracovat s konkrétním papírem se štítky:** Systém bude umožňovat uživateli práci s papírem a to konkrétně vytvářet nový papír, modifikovat, zobrazit informace o daném papíru a vykreslení. Při práci s papírem musí uživatel uvést tyto údaje.

  - Počet štítků na řádek
  - Počet štítků na sloupec
  - Šířka štítku
  - Výška štítku
  - Vykreslení daného QR kódu
  - Uvedení místa vložení QR kódu

- **Pracovat s konkrétní obálkou** Systém bude umožňovat uživateli práci s obálkou a to konktrétně vytvořit novou obálku, modifikovat, zobrazit informace a vykreslení. Při práci s obálkou musí uživatel uvést tyto údaje.

  - Název obálky
  - Šířka obálky
  - Výška obálky
  - Vykreslení QR kódu pro adresu a odesílatele

- **Vyřadit políčko:** Systém bude umožňovat uživateli vyřadit neexistují políčka na papíře v případě nalepovacích štítků.
- **Zobrazení odesílatelů:** Systém bude umožňovat uživateli zobrazení papíru s počtem odesílatelů korespondující s počtem adresátů.
- **Zobrazení archu:** Systém bude umožňovat uživateli zobrazení poštovního archu s předepsanými adresami. Tento podací arch slouží k potvrzení podaní zásilky.

2.2.4 Další funkcionalita

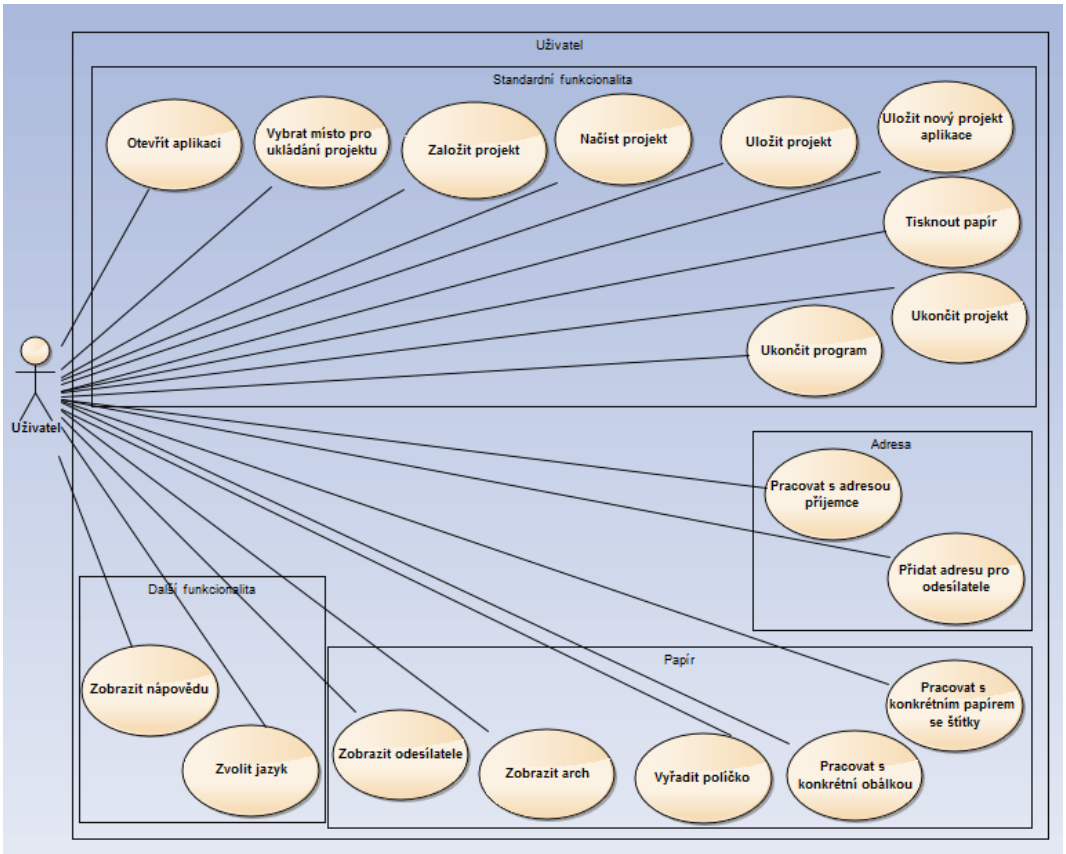
- **Zvolit jazyk:** Systém bude umožnovat uživateli výběr jazyka.
- **Zobrazit nápovědu:** Systém bude umožňovat uživateli zobrazení nápovědy k jednotlivým položkám aplikace pomocí tooltipu, který při umístění kurzorem myši na danou komponentu zobrazí popisek.

2.3 Nefunkční požadavky

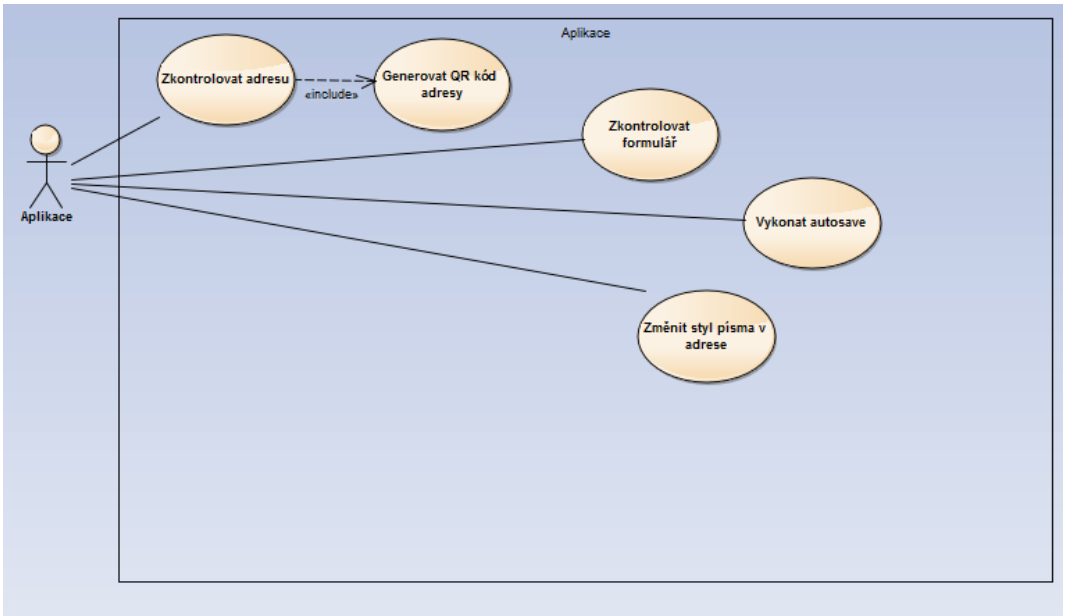
Nefunkční požadavky jsou omezení služeb nebo funkcí, které systém poskytuje a netýkají se konkrétních služeb, které systém poskytuje svým uživatelům.

- **Persistence:** Systém bude umožňovat ukládání adres a pamatování místa s projekty.
- **Platformní nezávislost:** Systém bude umožňovat podporu různých operačních systémů.
- **Grafické rozhraní:** Systém bude poskytovat GUI.
- **Programovací jazyk:** Systém bude implementovaný v jazyce Java.

2.4 Případy užití



Obrázek 2.1. Případy užití pro uživatele



Obrázek 2.2. Případy užití pro aplikaci

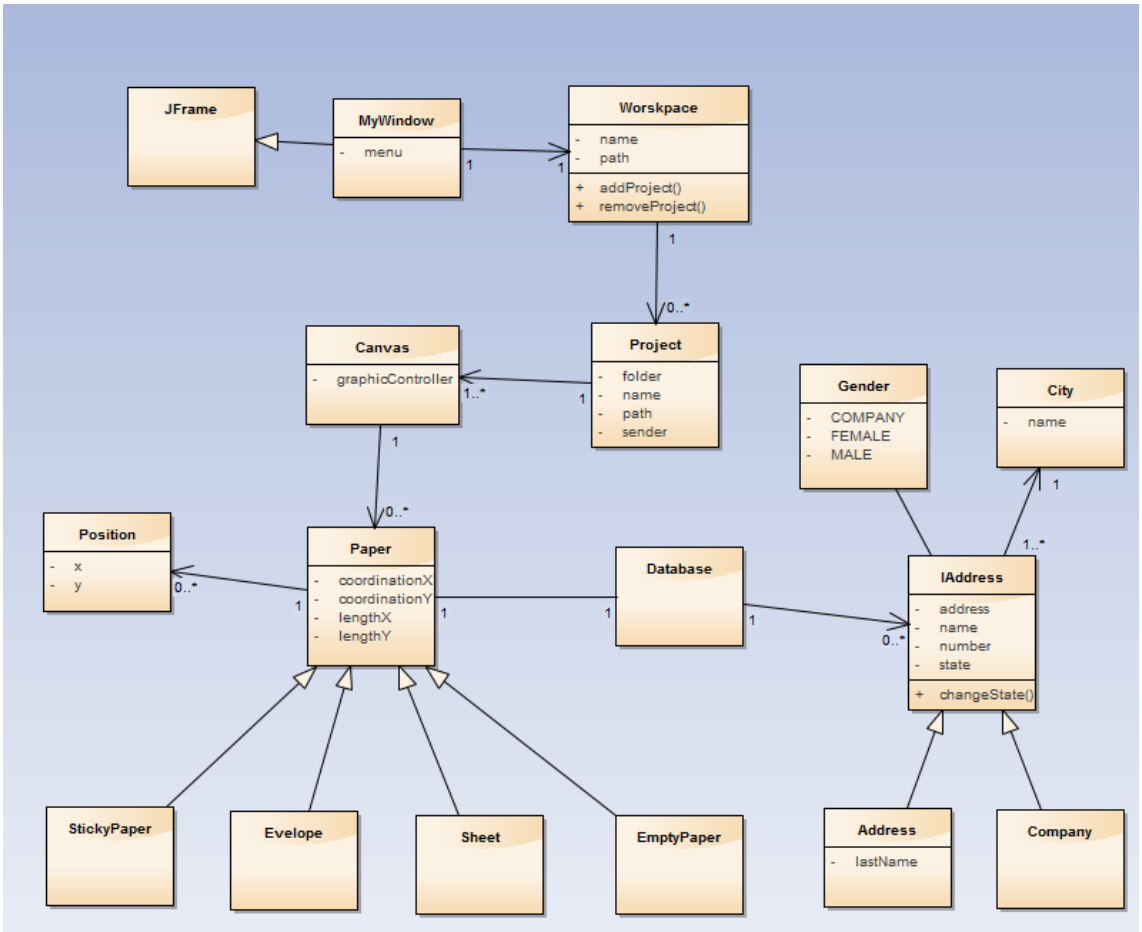
Pro přehled zde uvádím případy užití v diagramu, které jsou metodou pro zachycení funkčních požadavků na systém. Tento diagram nám ukazuje všechny možné akce, které může uživatel nebo také aplikace provádět s programem 2.1 2.2.

Kapitola 3
Návrh

V této kapitole nastíním rozložení všech komponent a funkcionalitu aplikace pomocí základních diagramů UML. Při vytváření návrhu jsem dbal na to, aby program byl vytvořený nejjednodušším způsobem. Diagramy týkající se vnitřního rozložení objektů jsem ponechal v anglickém jazyce, protože tímto jazykem je aplikace naprogramovaná. Diagramy aktivit jsou v českém jazyce.

UML je soubor grafických notací, který se používá při vývoji softwaru. Nejčastější způsob použití UML je použití pro tvorbu náčrtku. Vývojaři v tomto pojetí používají UML k usnadnění komunikace o některých aspektech systému. Návrh celého systému přes UML může obsahovat všechny detaily, nebo se může soustředit jen na určitou oblast. Text je převzatý z knihy [2]

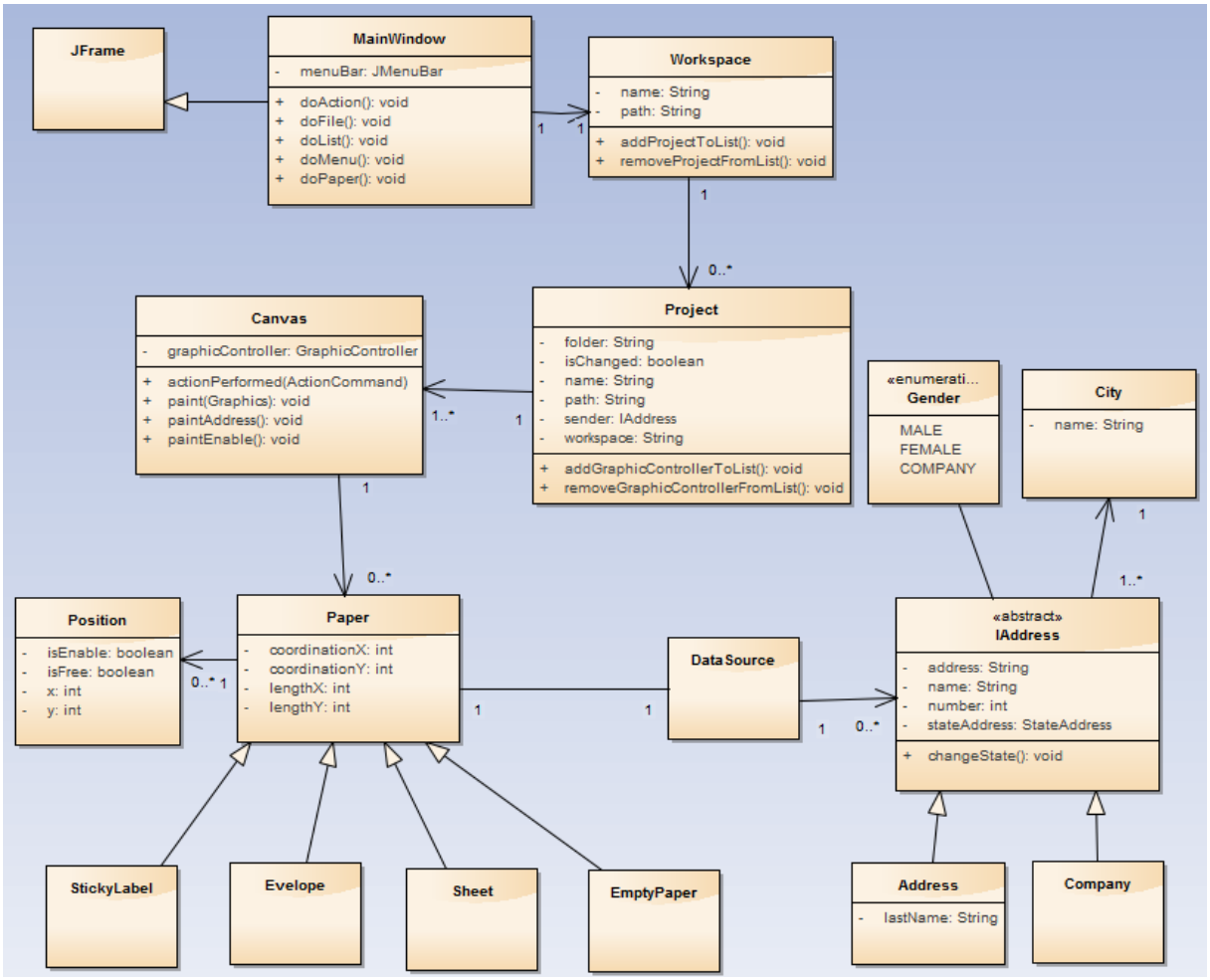
3.1 Analytický diagram



Obrázek 3.1. Analytický diagram

Analytický diagram 3.1 představuje pouze malý náznak celé aplikace. Můžeme zde vidět, že zde nejsou uvedé typy atributů a přesně popsané metody dané třídy. Je to pouze základní pohled na danou problematiku. Blíže tento diagram popisuji u Diagramu tříd.

3.2 Diagram tříd



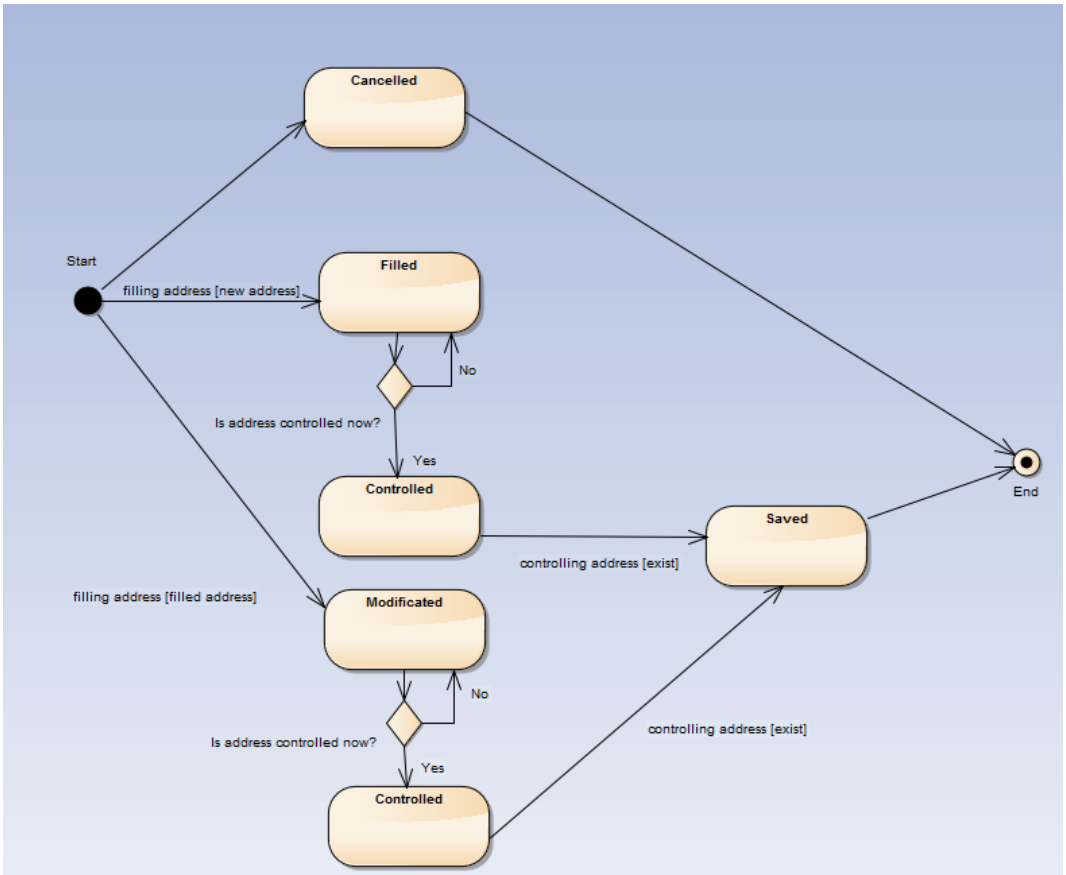
Obrázek 3.2. Diagram tříd

Diagram tříd 3.2 popisuje typy objektů v systému a různé druhy statických vztahů, které mezi nimi existují. Diagramy tříd ukazují i vlastnosti a operace třídy a také omezení týkající se způsobu, jakým jsou objekty spojovány.

Tímto diagramem představuji hlavní jádro celé aplikace za běhu založeného projektu. Dalo by se říct, že je to rozšíření analytického modelu, kde už můžeme vidět typy všech atributů a názvy metod, které třída obsahuje. Na začátku můžeme vidět objekt hlavního okna celé aplikace MainWindow, které dědí třídu JFrame pro zobrazení. Mezi hlavním oknem a projektem se nachází třída Workspace, která představuje prostředníka mezi GUI a logikou celého programu. Podle návrhu by nebylo dobré, aby si GUI udržovalo instanci již vytvořených projektů, na kterých se vykonává nějaká operace, proto jsem zvolil toto řešení. V této třídě je uchován název a cesta ke složce, ve které právě pracujeme. Je zde implementován list všech projektů, To můžeme zjistit tím, že v diagramu je vidět vztah mezi pracovním místem a danými projekty 1:N. Dále můžeme vidět vztah

mezi projektem a grafickým kontrolerem 1:4. To je z toho důvodu, že aplikace může mít na výběr ze čtyř různých papírů a to jsou nalepovací štítky, obálka, poštovní arch a v budoucnosti také prázdný papír. Každý z těchto papírů dědí třídu PaperView, která obsahuje společné atributy pro všechny papíry. Třída PaperView zde reprezentuje pouze jeden papír a tedy grafickému kontroleru připadá vztah 1:N. Na všechny papíry se bude muset vložit nějakým způsobem adresy pro vykreslení. Třída IAddress představuje rozhraní pro daný typ adresy, protože ji můžeme rozdělit na adresu jednoho příjemce a na adresu společnosti. Obsahuje také enum pro zobrazení pohlaví a třídu City reprezentující město. Pro vykreslení adresy na danou pozici jsem zvolil další třídu Position, která reprezentuje souřadnice vykreslení dané adresy.

3.3 Diagram stavů



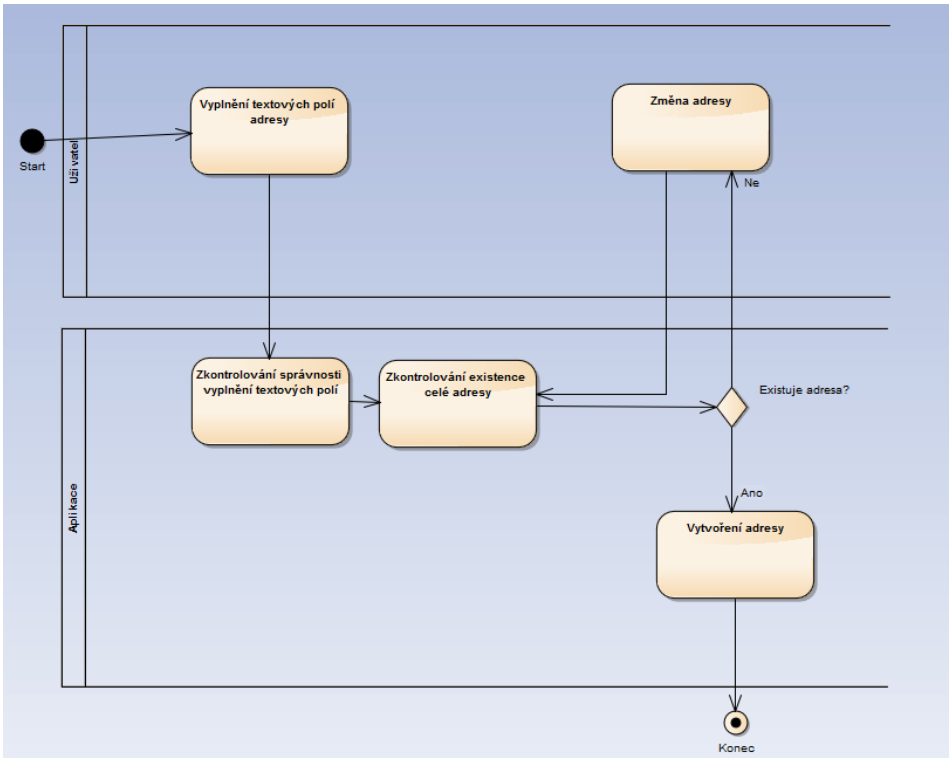
Obrázek 3.3. Diagram stavů

Diagram stavů 3.3 je známou technikou používanou k popisu chování systému. Tímto diagramem představuji aktuální stav, ve kterém se adresa nachází. Rozhodl jsem se použít toto řešení, protože je v aplikaci implementována kontrola adresy přes Open street map. V případě, že uživatel zadal nesprávnou nebo neúplnou adresu, může dojít k problému při hledaní daného místa. Při napsání adresy musí být všechny položky vyplněné a pak adresa bude ve stavu Filled, po úspěšném zkontrolování ve stavu Controlled a při vykreslení adresy ve stavu Saved. Dále pak můžeme adresu buď modifikovat nebo smazat. Fáze pro modifikaci je stejná, ale místo stavu Filled obsahuje stav Modificate.

3.4 Diagram aktivit

Diagram aktivit nám ukazuje postup při vykonání nějaké akce uživatelem. Tento diagram je technikou určenou k popisu a toku práce v systému. Může také zachytit paralelní chování. Uvádím zde všechny způsoby, jak může uživatel s aplikací spolupracovat.

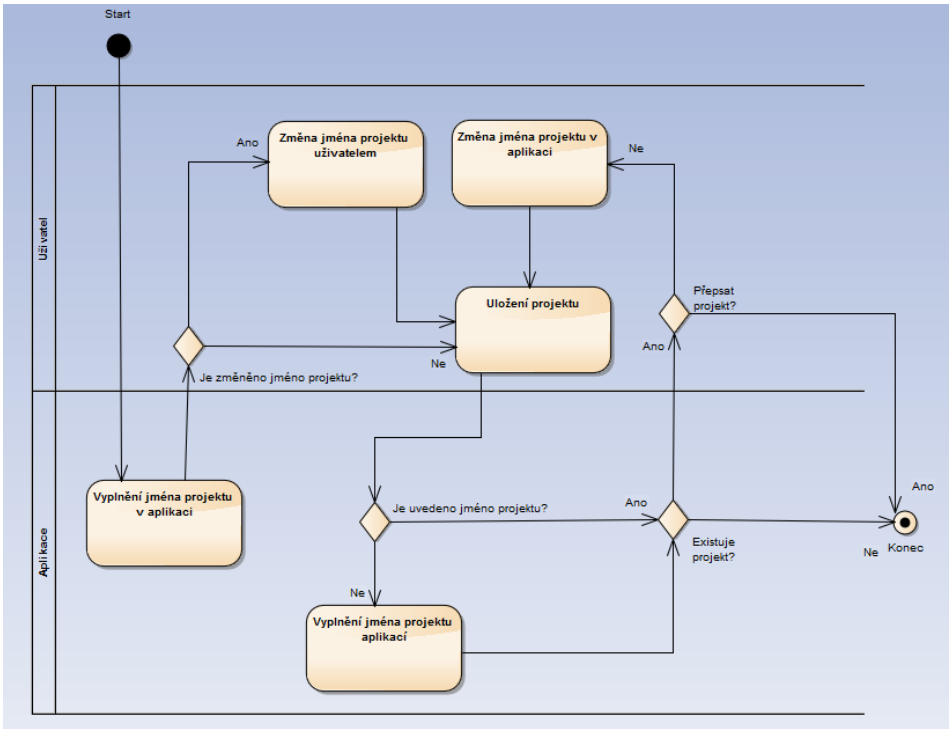
3.4.1 Založení adresy



Obrázek 3.4. Založení adresy

Uživatel vyplní všechny potřebné údaje pro vytvoření nové adresy 3.4. Kontrola správnosti vyplnění textových polí je popsáno ve Vyplnění formuláře. Pak musí uživatel adresu zkontrolovat, jestli je adresa platná. Pokud je adresa platná, uloží se a vykreslí se na papír. Pokud ne, uživatel musí změnit údaje a opět zkontrolovat.

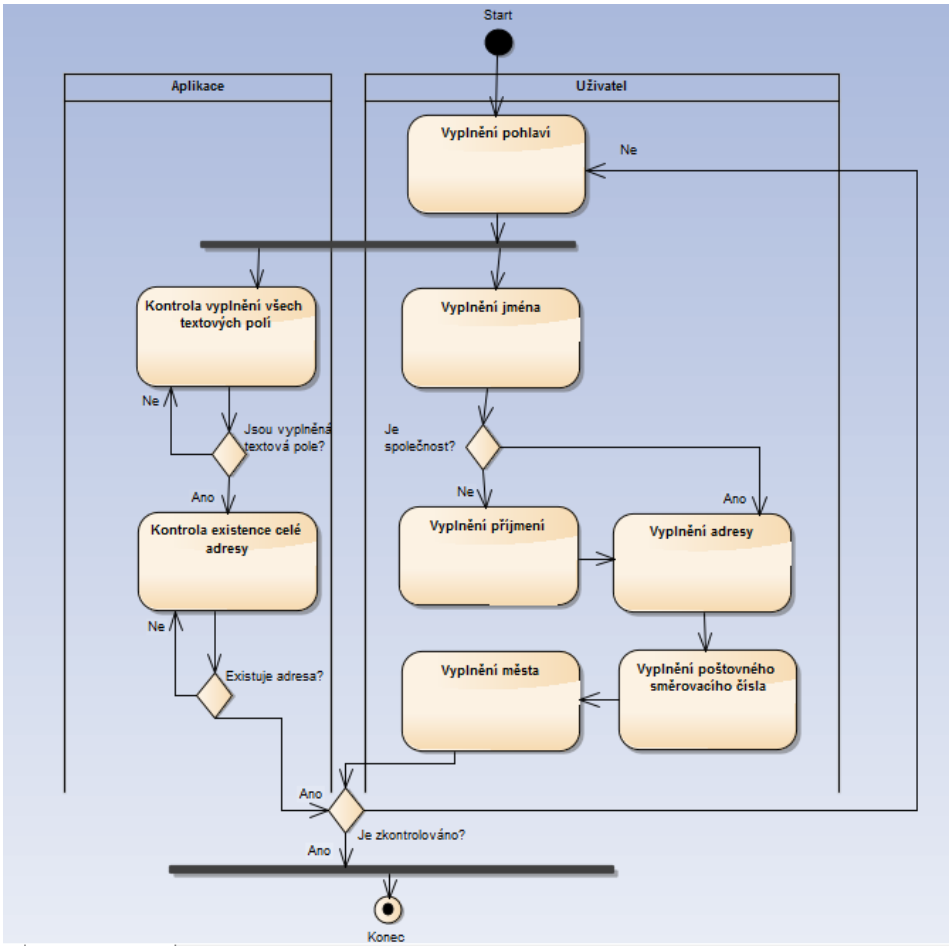
3.4.2 Uložení projektu



Obrázek 3.5. Uložení projektu

Uživatel vybere v menu položku New 3.5. Vyplní jméno nového projektu, zvolí papíry, které má nový projekt obsahovat, vybere formát všech papírů, které zvolil a dále vybere, jestli bude chtít tisknout odesílatele. Pokud uživatel nevyplnil jméno projektu, jméno projektu se bude shodovat s aktuálním datumem založení tohoto projektu. Po potvrzení všech vybraných možnosti si aplikace zkontroluje, jestli projekt už existuje. Pokud ano, aplikace se zeptá uživatele, jestli chce projekt přepsat. Pokud ne, vytvoří se nová složka s názvem projektu a v této složce se vytvoří MDB soubor, kam se budou ukládat zapsané adresy. V případě přepsání projektu se obsah MDB souboru pouze smaže. Blíže tuto akci popisuji v kapitole o implementaci.

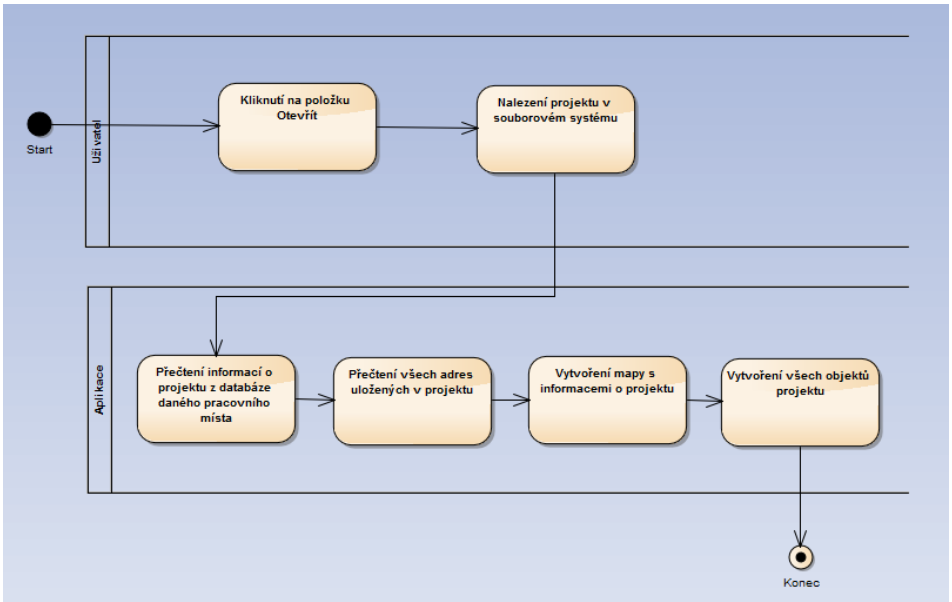
3.4.3 Vyplnění formuláře



Obrázek 3.6. Vyplnění formuláře

Uživatel nejprve vybere pohlaví nebo společnost 3.6. V tento moment se začnou provádět dvě akce najednou. Aplikace začne hned kontrolovat, jestli jsou všechna textová pole vyplněna a uživatel může vyplňovat udáje o příjemci. V případě, že aplikace zjistí, že jsou všechna textová pole správně vyplněna, zpřístupní uživateli možnost pro zkontrolování a založení adresy.

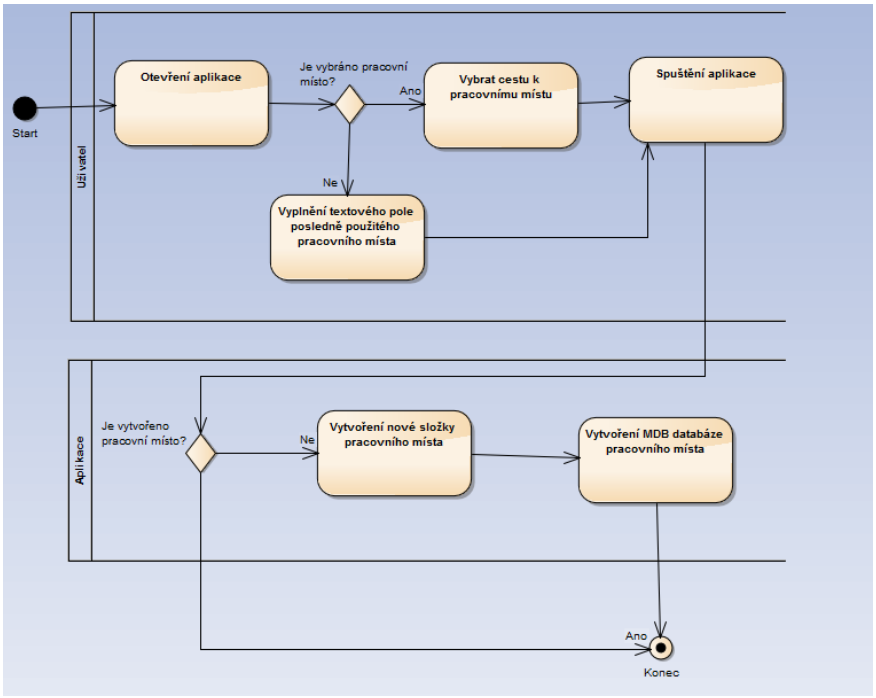
3.4.4 Načtení projektu



Obrázek 3.7. Načtení projektu

Uživatel zvolí načtení nového projektu 3.7. V tento moment se mu otevře dialog s rychlým hledáním v systému a vybere si MDB soubor v projektu, který chce právě načíst. Následně aplikace načte z databáze pracovního místa, kde je projekt uložen, všechna data, která patří k tomuto projektu. Aplikace také přečte všechny adresy uložené v MDB souboru. Vytvoří se mapa se všemi informaci o tomto projektu a pošle se dále ke zpracování, kdy se vytvoří všechny potřebné objekty k načtení projektu.

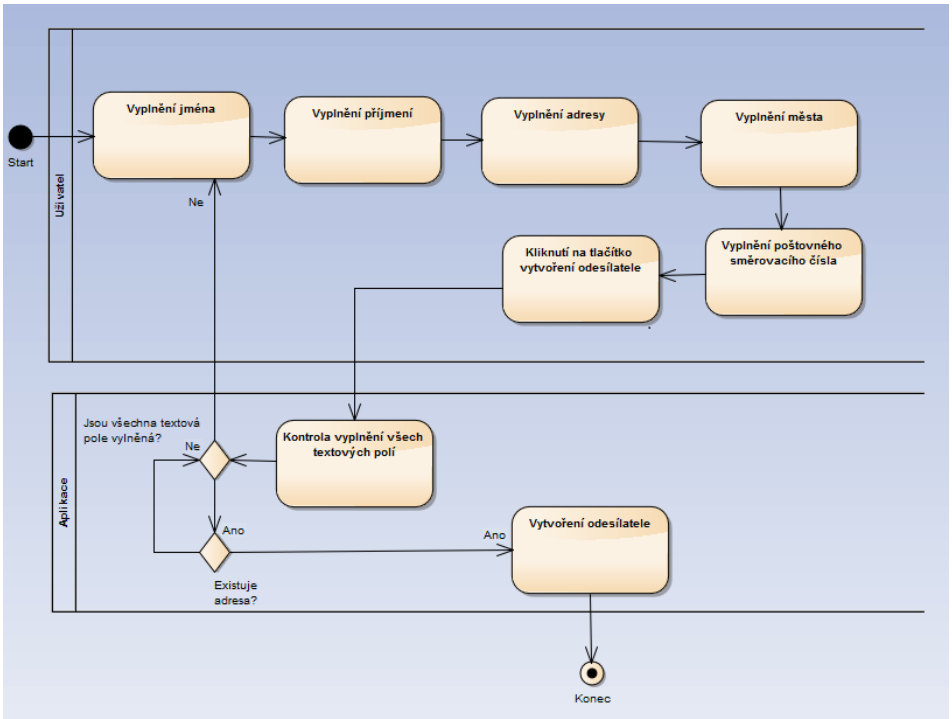
3.4.5 Založení pracovního místa aplikace



Obrázek 3.8. Založení pracovního místa aplikace

Uživatel napíše do textového pole cestu ke složce, ve které chce pracovat 3.8. Může si také vybrat volbu rychlého hledání v systému. Pokud uživatel textové pole nevyplní, zobrazí se mu hláška o nevyplnění cesty ke složce. V případě správného vyplnění textového pole aplikace zjistí, jestli dané pracovní místo existuje. Pokud ano, objeví se hlavní okno aplikace. Pokud ne, založí se složka s pracovním místem a v ní se vytvoří MDB databáze, kam se budou ukládat data o projektech. Práci s databází popisuji v kapitole o databázi.

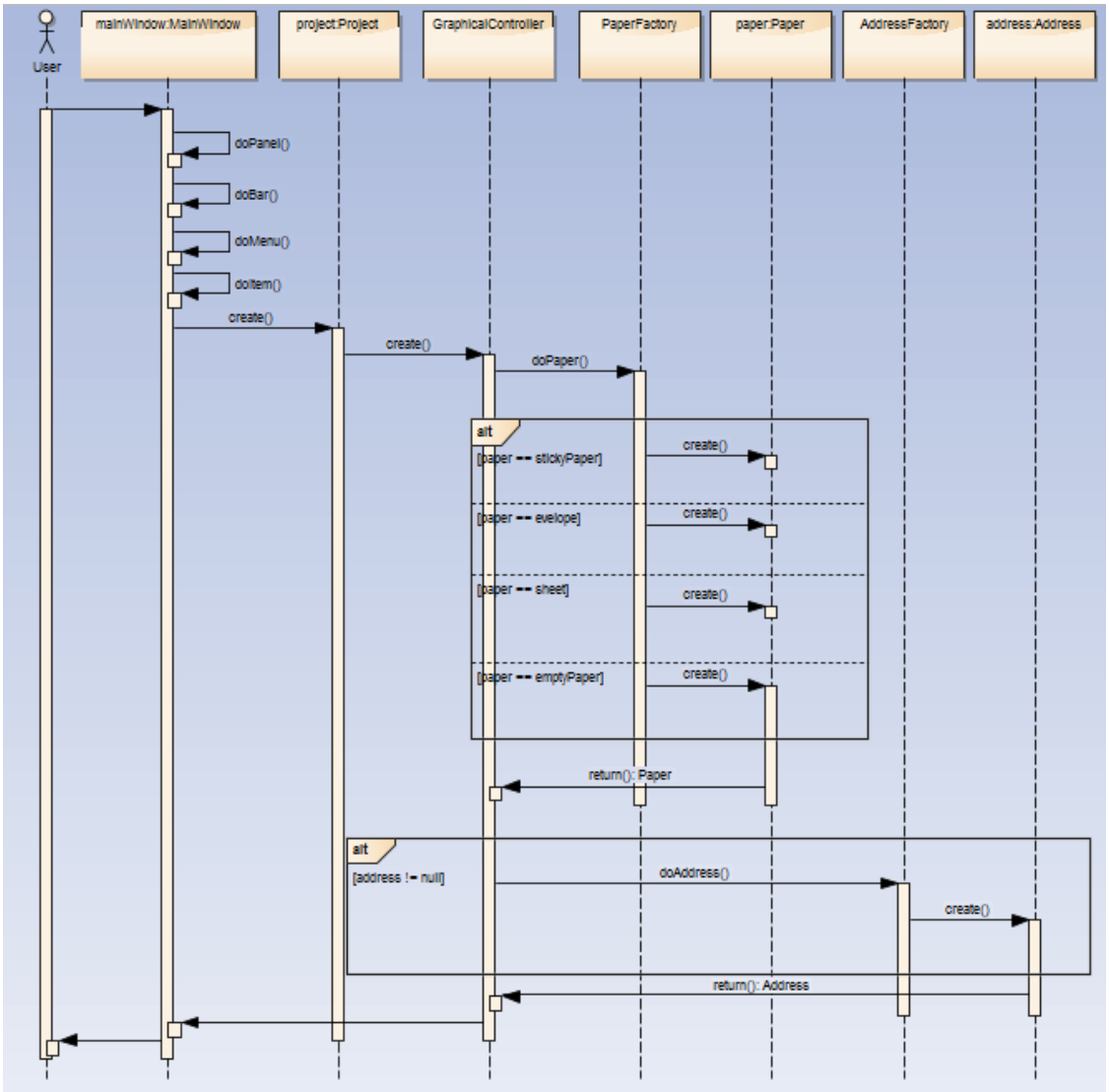
3.4.6 Vytvoření odesílatele



Obrázek 3.9. Vytvoření odesílatele

Pro vytisknutí poštovního archu je potřeba, aby byl uveden také odesílatel 3.9. Tuto akci může uživatel provést při založení nového projektu nebo během editace již existujícího projektu. Uživatel vyplní všechna textová pole o odesílateli. Program si zkontroluje vyplněné údaje a vytvoří odesílatele.

3.5 Sekvenční diagram



Obrázek 3.10. Sekvenční diagram

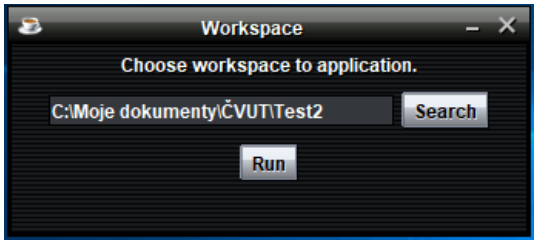
Sekvenční diagram typicky zachycuje chování jednoho scénáře 3.10. Tento diagram nám zde zobrazuje postup komunikace objektů při zakládání projektu a zapsání adresy.

Kapitola 4
Implementace

V této kapitole představím jednotlivé grafické prvky použité v aplikaci a jejich kódy pro vytvoření. V závěru kapitoly uvedu pár zajímavostí, které jsem použil k vytvoření aplikace.

4.1 Grafické uživatelské rozhraní aplikace (GUI)

4.1.1 Workspace



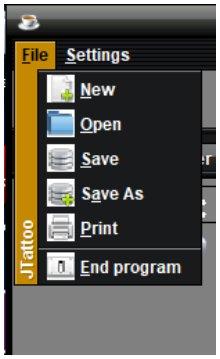
Obrázek 4.1. Workspace

Při spuštění aplikace se vždy zobrazí okno, kde si uživatel vybere, kam chce v tento moment všechny nově založené projekty uložit 4.1. Je zde zobrazena posledně použitá cesta, aby tuto cestu po každém spuštění nemusel uživatel hledat. Vedle textového pole pro cestu ke kořenové složce se nachazí tlačítko Search, které umožní uživateli rychlé hledání v jeho souborovém systému. Uživatel si tuto cestu může také napsat ručně, protože je zde naprogramováno ošetření neexistujících složek v počítači. Po kliknutí na tlačítko Run si program zkontroluje, jestli všechny složky uvedené v kořenové složce existují. Pokud ne, program všechny neexistující složky vytvoří a na konci založí soubor MDB, kam se budou ukládat všechny projekty v tomto pracovním místu. Další věc, kterou systém udělá je ta, že do souboru configuration.properties uloží právě zvolenou cestu. Po opětovném spuštění aplikace zde bude uvedena opět poslední použitá cesta.

4.1.2 Hlavní okno

Obrázek 4.2. Prázdné okno

Obrázek 4.3. Okno s otevřeným projektem



Obrázek 4.4. Soubor

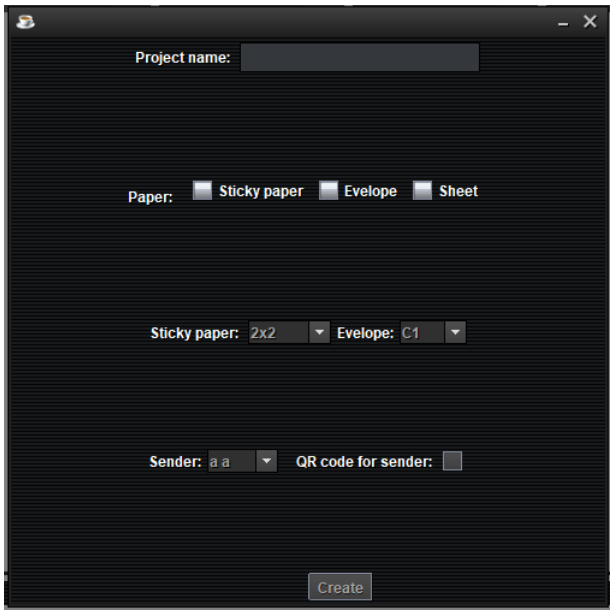
Obrázek 4.5. Nastavení

Po zvolení místa uložení projektu se uživateli objeví hlavní okno celé aplikace 4.2. Pro návrh vzhledu jsem použil javovský styl zvaný JTattoo [1] volně dostupný pro všechny vývojaře, který je ovšem trochu upravený, protože pozadní aplikace je obarveno černou barvou a pokud si otevřete jiné okno aplikace, názvy a jednotlivé položky nejsou moc čitelné. Z toho důvodu jsem obarvil pozadí toho stylu na šedou barvu, aby informace nesplývaly s pozadím.

Na horním panelu se nachází lišta s položkami File a Settings. Položka File má standardní volby New, Open, Save, Save as, Print a End program 4.4. Položka Settings obsahuje položky Database a volby jazyka celé aplikace 4.5. Zatím aplikace obsahuje pouze český jazyk a anglický jazyk. Všechny tyto uvedené položky rozebírám dále. Zobrazení hlavního okna se ovšem mění podle toho, jestli jsou v aplikaci otevřené nějaké projekty. Pokud uživatel založí nový projekt nebo je nějaký projekt otevřený, program si vytvoří dané instance a vloží je do objektů zvané JTabbedPane 4.3. Můžeme vidět, že jeden projekt se skládá ze dvou objektů JTabbedPane, kde první značí název projektu a druhý zobrazuje všechny zvolené papíry. Po kliknutí na volbu End program v položce File nebo kliknutí na křížek aplikace program zkontroluje všechny dosud otevřené projekty a zjistí, jestli jsou všechny projekty uloženy. Pokud ano, aplikace se v pořádku ukončí, pokud ne, program zahlásí, že ne všechny otevřené programy jsou uloženy a dotáže se na uložení projektů. Když se uživatel rozhodne ukončit aplikaci, systém si poznamená všechny otevřené projekty a při opakoveném spuštění programu daného pracovního místa budou všechny projekty automaticky otevřeny.

[1] `http://www.jtattoo.net/`

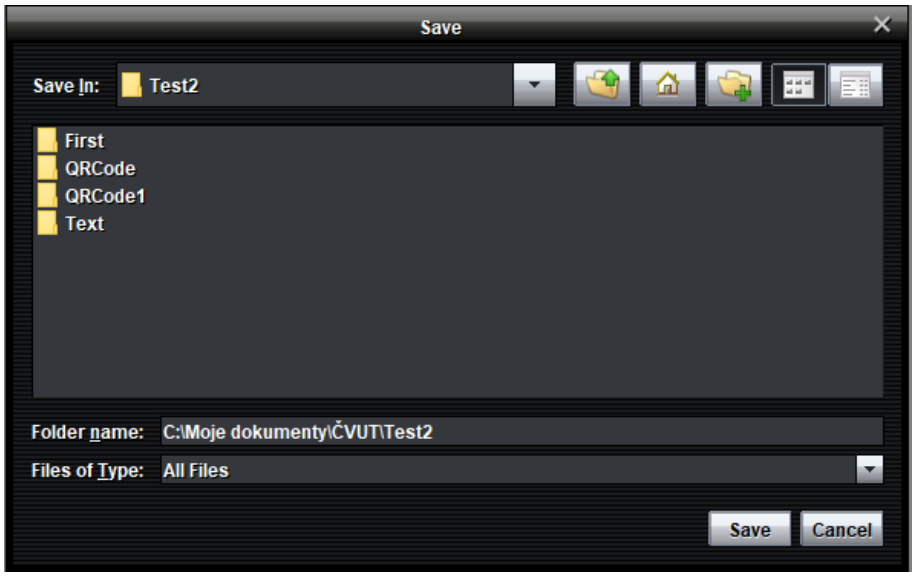
4.1.3 Nový projekt



Obrázek 4.6. Nový projekt

Při volbě nového projektu se uživateli otevře okno, kde si zvolí všechny potřebné informace, se kterými bude chtít v tomto projektu pracovat 4.6. Jak jsem psal v úvodu této práce, aplikace je tvořena tak, aby uživatele systém vedl a nemusel přemýšlet, co má v tuto chvíli dělat. Je vidět, že uživatel nemůže založit projekt, pokud nemá zvolený žadný papír. Po zvolení papíru se mu zpřístupní následují položky, kde si vybere typ papíru a v případě zvolení obálky také druh obálky. Následně si uživatel vybere daného odesílatele. Po kliknutí na tlačítko Create si program zkontrojuje, jestli je vše správně vyplněno a založí projekt. V případě nevyplnění názvu projektu, program pojmenuje projekt podle aktuálního data.

4.1.4 Uložení projektu



Obrázek 4.7. Uložení projektu

Obrázek 4.8. Autosave

V této aplikaci samozřejmě nesmí chybět volba uložení projektu 4.7. Zde je to zjednodušné tak, že pokud uživatel klikne na volbu Save, uloží se všechny projekty, které byly od posledního uložení změněny.

V aplikaci je také naprogramovaný autosave 4.8. Uživatel tedy nemusí myslet na to, že musí projekt ukládat. Autosave a stejně tak kontrola údajů ve formuláři by se dalo jednoduše naprogramovat přes vlákna zděděním třídy Thread. Tento způsob ovšem není zcela efektivní a použití toho způsobu může vést ke snížení výkonu a zvýšení procesorového času [3].

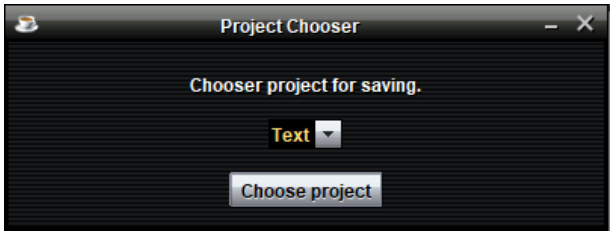
```
public class MainWindow extends AMainWindow implements ActionListener,
WindowListener, ComponentListener, MouseListener, MouseMotionListener{

private Workspace workspace = null;
    .
    .
     public MainWindow(String workspace){
         Autosave autosave = new Autosave(this.getWorkspace, this);
         timer.schedule(autosave, 1000, 5000);
     }
     .
     .
}
```

Obrázek 4.9. Spuštění autosavu

```
public class Autosave extends TimerTask{
.
.
     @Override
     public void run(){
     }
.
.
}
```

Obrázek 4.10. Autosave

4.1.5 Uložení nového projektu aplikace

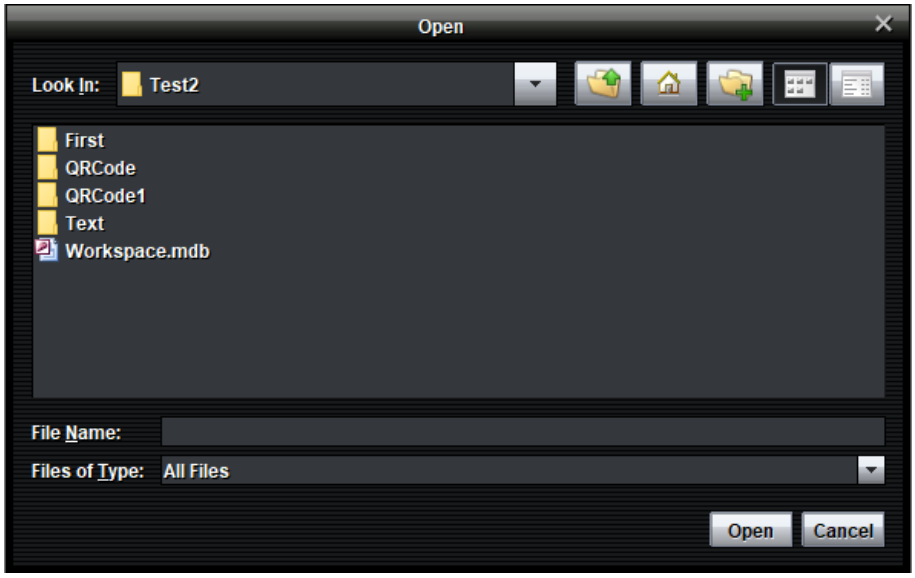


Obrázek 4.11. Uložení nového projektu aplikace

Uživatel se může rozhodnout, že by si stávající stav projektu chtěl někam uložit a pokračovat v kopii a následně se vrátit k původnímu projektu. Z toho důvodu zde existuje

možnost Save as 4.11. Jinak by zde tato možnost byla zbytečná, protože program ukládá projekt automaticky z okna New. Při přeuložení nastává problém, pokud v aplikaci je otevřeno více projektů. Program by se dal samozřejmě naprogramovat tím způsobem, že se přeuloží právě aktivní projekt nebo tedy aktivní JTabbedPane. Uživatel si v tomto případě nemusí být jistý, jestli ukládá správný projekt a z tohoto důvodu se při volbě možnosti Save as, kdy je v aplikaci otevřeno více než jeden projekt, otevře okno, kde si uživatel vybere, který projekt chce vlastně přeuložit.

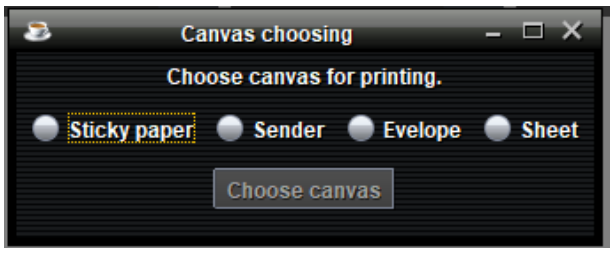
4.1.6 Načtení projektu



Obrázek 4.12. Načtení projektu

Při volbě možnosti Open se uživateli otevře okno, kde se může vybrat projekt z daného pracovního místa, který chce pravě otevřít 4.12. Po zvolení konkrétního projektu si program v dané databázi najde všechny informace o projektu zahrnující výskyt daného typu papíru, jestli je zde QR kód a odesílatel. Pokud se tedy stane, že uživatel bude chtít otevřít projekt, který už je otevřený, program mu vypíše hlášku o otevřeném projektu.

4.1.7 Tisk



Obrázek 4.13. Tisk

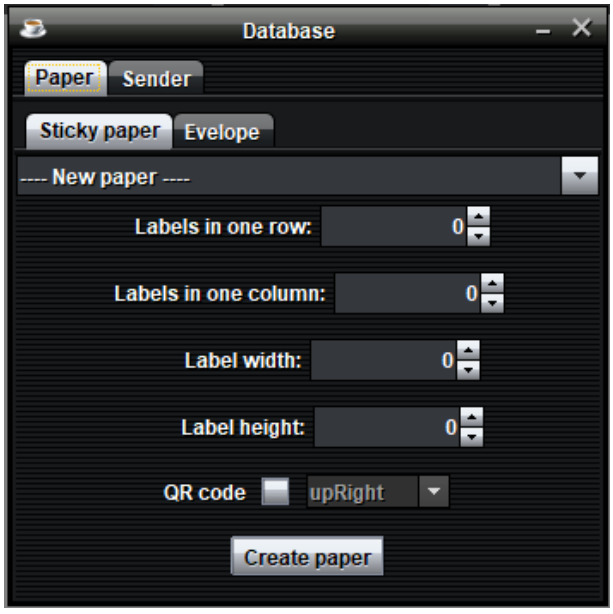
Program samozřejmě umí také tisknout jednotlivé papíry. Při zvolení volby Print se uživateli nejprve objeví okno o výběru tisku daného papíru 4.13. Tiskárna sama neumí rozlišit, o jaký typ papíru se jedná a tak si toto musí zařídit sám uživatel. Po zvolení daného papíru se uživateli objeví tiskové okno, kde si může zvolit, kolik papírů chce

tisknout a následně klikne na tlačítko Tisk. Kód pro tisk je upravený podle potřeby aplikace [4].

```
public class paperChooserWithSevice{
    PrinterJob pjob = PrinterJob.getPrinterJob();
    PageFormat pageFormat = pjob.defaultPage();
    Paper paper = new Paper();
    paper.setImageableArea(0,0, pageFormat.getWidth(), paper.getHeight());
    .
    .
    PrintService service = PrintServiceLookup.lookupDefaultPrintService();
    .
    .
    if(pjob.printDialog()){
        try{
            pjob.print();
       }catch(PrinterException ex){
           Logger.getLogger(MainWindow.class.getName())
.log(Level.SEVERE, null, ex);
       }
    }
}
```

Obrázek 4.14. Tisk

4.1.8 Vložení nového papíru do databáze

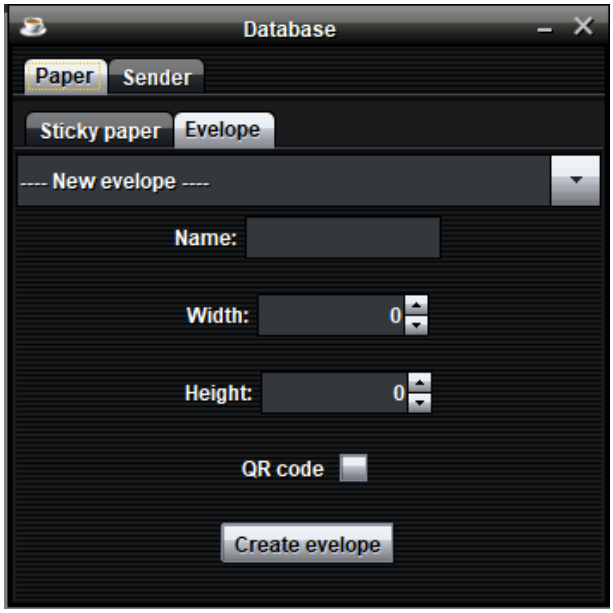


Obrázek 4.15. Vložení nového papíru do databáze

Při volbě Database v položce Settings se uživateli objeví okno, které spolupracuje s databází 4.15. Jedna z možností je zde založení vlastního papíru. Uživatel zadá počet sloupců a počet řádků, kolik má papír obsahovat štítků, šířku a výšku jednoho štítku. Poslední volba je zde možnost tisku adresy s QR kódem. Tento vstup od uživatele je samozřejmě ošetřen tím způsobem, že pokud zadá neplatné údaje nebo jsou jednotlivé rozměry mimo výšku a šířku papíru, program vypíše uživateli danou hlašku o nesprávném vyplnění údajů. Je zde také možnost úpravy papíru. V rozbalovací liště si uživatel

najde daný papír, který si chce upravit a program mu sám předvyplní aktuální informace o papíru. Po úpravě informací o papíru se opět provede kontrola a pokud rozměry štítků souhlasí s papírem, papír se úspěšně modifikuje v databázi.

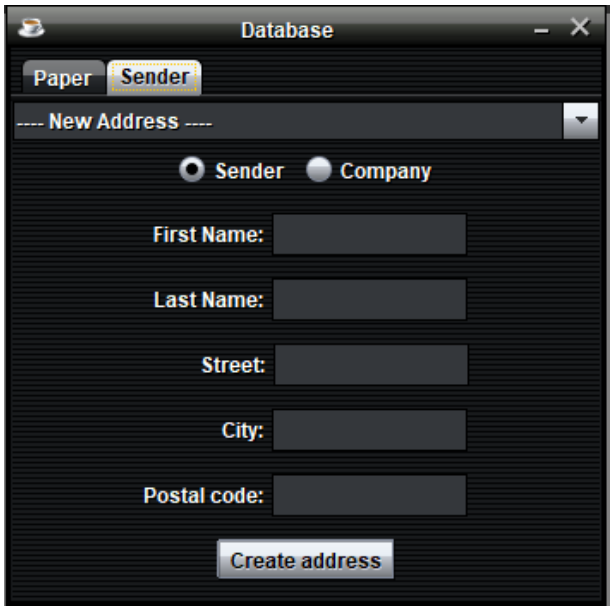
4.1.9 Vložení nové obálky do databáze



Obrázek 4.16. Vložení nové obálky do databáze

Možnost přidání a úpravy obálky funguje stejně 4.16.

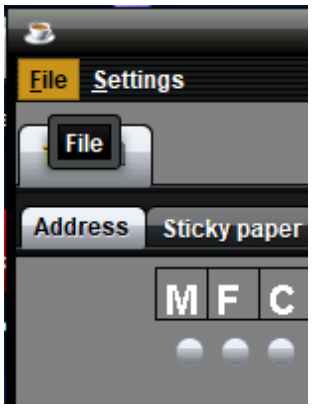
4.1.10 Vložení nového odesílatele do databáze



Obrázek 4.17. Vložení nového odesílatele do databáze

Možnost založení odesílatele funguje také na stejném principu, ale je zde navíc možnost kontroly adresy přes mapy na internetu 4.17.

4.1.11 Nápověda



Obrázek 4.18. Nápověda

Při umístění kurzoru myši na danou komponentu v programu se zobrazí napověda, která má za účel vysvětlení akce při kliknutí 4.18.

```
public class AddressCanvas extends JPanel
implements MouseMotionListener(){
.
.
public void mouseMoved(MouseEvents me){
 this.setToolTipText(LanguageUtil.findWord("File"));
}
.
.
}
```

Obrázek 4.19. Nastavení tooltipu na plátno

4.1.12 Změna jazyka

Obrázek 4.20. Změna jazyka

Uživatel si může měnit jazyk aplikace ve volbě Language v položce Settings 4.20.

4.1.13 Formulář

Obrázek 4.21. Formulář

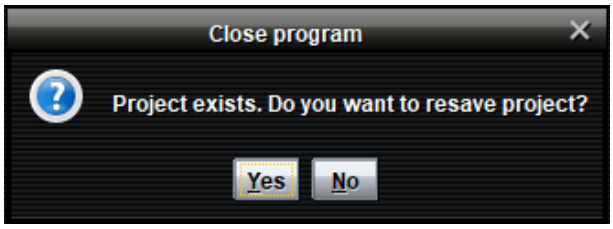
Celý panel, který představuje formulář, je nejzásadnější componenta v celé aplikaci 4.21. Při založení adresy musíme uvést pohlaví, jméno, adresu, město a poštovní směrovací číslo. Adresa může také odkazovat na firmu. V tomto případě se zvolí místo pohlaví společnost a textové pole pro příjmení zmizí. Existuje zde také postupné zakládání adresy. Tedy nemůžeme založit adresu, pokud ji nezkontrolujeme její existenci. Adresu nemůžeme ani zkontrolovat, pokud nevyplníme všechny údaje a nemůžeme vyplnit všechny údaje, pokud nevybereme pohlaví nebo společnost. Tím se zamezí případným chybám a uživatele tak aplikace vede ke správnému vyplnění adresy.

Co se týká správného vyplnění textových polí, je zde naprogramováno v jakém formátu se mají data vyskytovat. To znamená, že v poli pro jméno a příjmení se musí vyskytovat pouze řetězec písmen, pole pro ulici se musí skládat z řetězce a čísla popisného, popřípadě čísla s lomítkem, pole pro město obsahuje pouze řetězec a pole pro poštovní směrovací číslo musí být celé číslo a zaroveň pětimístné.

Funkcionalitou, kterou jsem vylepšil tento formulář je ta, že při zadání adresy program začne hledat všechna města a k ním patřící poštovné směrovací čísla. Pokud najde alespoň jeden výsledek, předvyplní textové pole pro město a PSČ. Je ovšem možné, že se zadaná adresa může vyskytovat několikrát, pak si může uživatel z rozbalovacího textového pole vybrat konkrétní město. Může se ovšem stát, že mapy požadované město nenajdou. V tomto případě si uživatel všechny položky vyplní sám. Pokud je vše v tomto formátu vyplněno, uživateli se zpřístupní možnost Check. Po kliknutí na tuto možnost program ještě jednou zkontroluje správnost adresy podle map [1] na internetu.

Je zde uvedena ještě jedna kontrola a to kontrola správnosti města a poštovního směrovacího čísla. V případě nalezení dané adresy v mapách se uživateli zpřístupní možnost Create, která adresu vloží do všech zvolených papírů v projektu. K vytvořeným adresám se také generuje QR kód s odkazem na mapy daného místa.

4.1.14 Výstražná okna



Obrázek 4.22. Uložení již existujícho projektu

Pokud se v nějaké časti programu stane situace, která by mohla způsobit problém, objeví se uživateli výstražné okno, které zamezí vzniku těmto chybám 4.22. Patří mezi ně například otevření už otevřeného projektu, přepsání projektu, nekorektnost adresy, nesprávné vyplnění textových polí, nesprávné rozměry štítků a obálek při vložení nových údajů do databáze a tak dále.

[1] `https://cs.wikipedia.org/wiki/OpenStreetMap`

```
public class Message extends JFrame{

    final static int[] MESSAGE_LIST = new int[]
    {JOptionPane.INFORMATION_MESSAGE, JOptionPane.QUESTION_MESSAGE,
    JOptionPane.WARNING_MESSAGE, JOptionPane.ERROR_MESSAGE};

    public static int showMessageOfApplication(String message,
    int messageIndex){
        int answerType = 10;
        JFrame frame = new JFrame();
        if(messageIndex == 1){
            answerType = JOptionPane.showConfirmDialog(frame, message,
           "Close program", JOptionPane.YES_NO_OPTION);
        }else{
            JOptionPane.showMessageDialog(frame, message,
 "Message", MESSAGE_LIST[messageIndex]);
        }

        return answerType;
    }
}
```

Obrázek 4.23. Implementace výstražného okna v aplikaci

4.2 Zajímavosti

V této části kapitoly vystihnu některé zajímosti, které jsem při vytváření programu použil. Konkrétně se jedná o návrhové vzory. Při objektově orientovaném programování musíme řešit programátorské postupy, které zajistí lepší čitelnost kódu a také zlepší práce s celým programem samotným. Objektově orientované návrhové vzory ukazují vztahy a interakce mezi třídami a objekty. Základním kamenem pro návrhové vzory je kniha Design Patterns: Elements of Reusable Object-Oriented Software, jejímiž autory jsou Erich Gamma, Richard Helm, Ralph Johnson a John Vlissides. Těmto čtyřem zakladatelům se také říká Gang of Four.

Základními typy návrhových vzorů jsou Creational Patterns, Structural Patterns a Behavioral Patterns. První typ řeší situaci s vytvářením objektu v systému. Popisuje postup při výběru nového objektu a zajištění správného počtu těchto objektů. Druhý typ se zaměřuje na uspořádání jednotlivých tříd a komponent v systému. Poslední typ se zajímá o chování celého systému. U tříd se využívá princip dědičnosti a u objektů jejich komunikace mezi sebou. Text je až na výjimky převzatý a částečně přeložný z [5] a [6].

4.2.1 Singleton

Singleton je návrhový vzor, který umožňuje globální přístup k instanci dané třídy. Jde o takový vzor, který od jednoho druhu objektu vytvoří pouze jednu instanci, kterou si uloží do proměnné. Nejjednodušší implementace vzoru obsahuje statickou metodu getInstance(), která volá kontruktor třídy a statickou proměnou, kam je uložena instance třídy. Nejznámější příklad použití Singletonu je databázové připojení.

```
public class DatabaseSpace implements IDatabaseSpace{

    private static DatabaseSpace databaseSpace;

    public static DatabaseSpace getInstance(){
       if(databaseSpace == null){
            databaseSpace = new DatabaseSpace();
        }
        return databaseSpace;
    }

}
```

Obrázek 4.24. Singleton

4.2.2 Factory

Účelem toho návrhového vzoru je vytvořit novou instanci nějakého objektu a vytvořenou instanci vrátit. Druh objektu a vlastnosti jsou dané přijatými parametry, případně i stavem objektu, který tovární metodu poskytuje. Tento návrhový vzor jsem v práci použil na vytváření jednotlivých modelových tříd. Tyto třídy můžeme vidět v diagramu tříd.

```
public class ProjectFactory{
public static Project createProject(String name, String workspaceName){
        Project project = new Project();
        project.setProjectName(name);
        project.setFolder(name);
        project.setWorkspace(workspaceName);
        project.setGraphicControllerList(new ArrayList⟨GraphicController⟩());
        return project;
    }
}
```

Obrázek 4.25. Factory

4.2.3 State

Návrhový vzor State je vhodné řešení v případě, kdy objekt během své existence mění své vnitřní chování tak, že nabývá různých stavů. Přičemž chování objektu se v různých stavech mění. V aplikaci jsem návrhový vzor State použil na jednotlivé objekty adres. Adresy mohou nabývat různých stavů například Zkontrolovaná, Vytvořená, Uložená, Modifikovaná.

```
public class IAddress{
public void doSave(){
            if(!(stateAddress instanceof StateControlled)){
                    Message.showMessageOfApplication("CheckAddress", 0);
             }else{
                getStateAddress().doSave(this);
            }

    }
}
```

Obrázek 4.26. Metoda doSave() v modelové třídě pro adresu

```
public class IAddress{
.
.
public void changeState(StateAddress stateAddress){
        this.setStateAddress(stateAddress);
    }
.
.
}
```

Obrázek 4.27. Metoda pro změnu stavu

```
public abstract class StateAddress {
    public void doFill(IAddress address){};
    public void doCheck(IAddress address, String address2, String path,
    String project, int positionLength){};
    public void doModificate(IAddress address){};
    public void doSave(IAddress address){};
    public void doCancel(IAddress address){};

    public void changeState(IAddress address, StateAddress stateAddress){
        address.changeState(stateAddress);
    }

}
```

Obrázek 4.28. Stav adresy

```
public class StateControlled extends StateAddress{
    .
    .
    public StateControlled(){

    }

    @Override
    public void doSave(IAddress address){
        changeState(address, new StateControlled());
    }
    .
    .
}
```

Obrázek 4.29. Změna stavu ze zkontrolované adresy na uloženou adresu

4.2.4 Builder

Builder je návrhový vzor popisující způsob vytváření složitých objektů z menších objektů. Obecně tento vzor zachycuje společnou kostru postupu a sníží se tak duplicita kódu. Tento vzor jsem v programu použil na vytváření projektu, kde každý projekt obsahuje dané typy papírů. Typů papírů je několik a pomocí tohoto návrhové vzoru postupně stavím celý projekt. Vytvoří se tedy jeden řídí objekt, který podle konkrétního stavitele předaného v paramteru funkce postaví daný objekt.

```
public interface IBuilder{
createGraphicControllerForPaper();
createPosition();
}
```

Obrázek 4.30. Rozhraní pro daného stavitele

```
public interface iDirector{
Project build(IBuilder builder, Project project,
 Map⟨String, String⟩ project information, int index);
}
```

Obrázek 4.31. Řídící třída pro stavitele

```
public class Builder implements IBuilder{

GraphicController graphicController;

public createGraphicControllerForPaper(){

}

public void createPosition(){

 }
}
```

Obrázek 4.32. Konkrétní stavitel

```
public class Director implements IDirector{

    public Project build(IBuilder builder, Project project,
    Map⟨String, String⟩ project information, int index){
        builder.createGraphicControllerForPaper()
        builder.createPositions();
    }
}
```

Obrázek 4.33. Konkrétní řídící třída pro stavitele

```
IDirector director = new Director();
director.build(builder, project, projectInformation, index);
```

Obrázek 4.34. Volání stavitele

4.2.5 Využití dědičnosti

Je samozřejmě jasné, že navržená okna mohou vypadat jinak, pokud by se do programu přidala možnost výběru typu rozložení komponent, kdyby se nějakému uživateli náhodou rozložení komponent nelíbilo. Z tohoto důvodu jsem pro každé okno vytvořil abstraktní třídu, která obsahuje potřebné proměnné a metody pro práci. Taková třída obsahuje i konkrétní abstraktní metody, které musí implementovat každá třída. Tuto třídu pak jen zdědí daná implementace okna.

```
public abstract class AWorkspaceWindow extends JFrame{
    .
    .
    void doAction(){}
    .
    .
}
```

Obrázek 4.35. Abstraktní třída pracovní místo

```
public class WorkspaceWindow extends AWorkspaceWindow
implements ActionListener{
    .
    .
    public void doAction(){
    }
    .
    .
}
```

Obrázek 4.36. Třída pro zobrazení pracovního místa

Při vývoji tohoto softwaru došlo také na implemetaci vlastních komponent. První příklad, který zde uvedu je situace, kdy uživatel bude chtít zavřít projekt, ve kterém skončil práci. Bylo by vhodné tedy nezavírat celou aplikaci, ale pouze tento projekt. Z toho důvodu jsem v aplikaci implementoval svoje vlastní tlačítko, které se chová stejně podle objektu JButton, ale vzhled je jiný. Třída tedy zdědila třídu JButton a přetížila se metoda na vykreslení tlačítka. Zděděním rozhraní ActionListener se provede vyjmutí objektu JTabbedPane daného projektu a zděděním rozhraní MouseListener se obarví tlačítko na barvu při přejetí myši.

```
public class Button extends JButton implements
ActionListner, MouseListener{
    .
    .
    @Override
    public void paintComponent(Graphics g){
        if(!isEntered){
            g.setColor(Color.BLACK);
        }else{
            g.setColor(Color.ORANGE);
            g.fillRect(0, 0, 10, 10);
            g.setColor(Color.BLACK);
            int k = 1;
        }
        g.drawLine(2, 2, 8, 8);
        g.drawLine(8, 2, 2, 8);
    }
    .
    .
}
```

Obrázek 4.37. Tlačítko pro zavření projektu

Kapitola 5
Databáze

Adres v projektu může být opravdu hodně a bylo by dobré mít zde také nějakou funkci pro uložení rozpracovaného projektu tak, jak je v běžných programech známo. Nejvhodnější způsob je založení nějaké databáze, která si bude uchovávat informace o daném pracovním místě a založených projektech. Pro tento typ ukládání jsem zvolil databázi MDB, která lze volně založit a pracovat s ní z programu MS Access. Jedná se o databázový software vytvářející databázové tabulky.

5.1 Jackcess

Jackcess [1] je java knihovna pro čtení a zápis do MS Access databáze. Poskytuje multiplatformní backend API povolující vývojářům pracovat s daty. Je to část projektu OpenHMS [2] od společnosti Health Market Science, která obsahuje kolekci další knihoven, API a nástrojů pro práci a manipulaci s daty pro vývoj profesionálních softwarů. Samotná knihovna Jackcess není aplikace a neposkytuje grafické uživatelské rozhraní. Je to knihovna pro vyvojáře, kteří ji používají pro vývoj aplikací v Javě.

5.2 Práce s databází aplikace

V této kapitole představím jednotlivé metody databáze, která spolupracuje s aplikací a využívá jednotlivých metod z knihovny Jackcess.

5.2.1 Připojení k databázi MDB

```
public void setDatabase(String name, String databasePath){
       try{
           if(!databaseMap.containsKey(name)){
                this.database = new DatabaseBuilder()
     .open(new File(databasePath));
                databaseMap.put(name, database);
           }else{
                this.database = databaseMap.get(name);
           }
       }catch(IOException e){}
   }
```

Obrázek 5.1. Připojení ke konkrétní databázi

[1] http://jackcess.sourceforge.net/
[2] http://openhms.sourceforge.net/

5.2.2 Vložení dat

```
public void insert(Map⟨String, Object⟩ dataMap) throws IOException{
        table.addRowFromMap(dataMap);
    }
```

Obrázek 5.2. Vložení dat

5.2.3 Čtení dat

```
public List⟨Row⟩ read(String key, String name) throws IOException{
        List⟨Row⟩ rowList = new ArrayList⟨Row⟩();
        for(Row row : table){
            if(!name.isEmpty()){
                 if(row.get(key).toString().equals(name)){
                     rowList.add(row);
                     return rowList;
                 }
            }else{
                rowList.add(row);
            }
        }
        return rowList;
    }
```

Obrázek 5.3. Čtení dat

5.2.4 Modifikace dat

```
public void update(Map⟨String, Object⟩ dataMap, String key,
String name2, int variable) throws IOException{
        Row row2 = read(key, name2).get(0);
        row2.putAll(dataMap);
        table.updateRow(row2);
    }
```

Obrázek 5.4. Modifikace dat

5.2.5 Mazání dat

```
public void delete(String name) throws IOException{
        List⟨? extends Column⟩ columnList = table.getColumns();
        List⟨Row⟩ rowList = null;
        if(!name.isEmpty()){
             rowList = read(columnList.get(0).getName(), name);
             table.deleteRow(rowList.get(0));
        }else{
            rowList = read("", "");
            for(Row row : rowList){
                table.deleteRow(row);
            }
        }
    }
```

Obrázek 5.5. Mazání dat

Kapitola 6
JUnit testy a integrační testy aplikace

Testování softwaru je klíčová součást vývoje. Dalo by se říct asi za nejdůležitější, protože při vývoji aplikace mohou nastat v softwaru problémy, které nejsou hned vidět a přesně proto je potřeba tyto testy psát. To, o co nám vlastně jde, je napsat automatický test pro určitou funkčnost nebo komponentu programu, který pak můžeme opakovaně spouštět. Existuje mnoho druhů testování softwaru, ale mezi nejzákladnější typy testování patří unit testy a integrační testy. Tyto testy dále popíšu podrobněji.

6.1 JUnit testy

JUnit testy slouží k testování menších jednotek zdrojového kódu. Naším cílem je vytvořit takové metody, které budou testovat jednotlivé metody zahrnuté v hotové aplikaci. Takové metody se většinou vyskytují ve speciálních třídách v jiném balíčku, než je samotná aplikace. Tyto testy programátorovi ověřují, že nová nebo změněná část kódu funguje, že její funkce odpovídá očekávání. Standard, kterým se implementují jednotkové testy, je knihovna JUnit nebo take TestNG. Každý test by měl testovat pouze jednu vlastnost nebo funkčnost testované komponenty a měly by pokrývat naprostou většinu funkčnosti aplikace.

Může se ovšem stát, že testovaná komponenta nebo metoda bude spolupracovat s jinou komponentou v jiné třídě, o které předpokládáme, že funguje v každém případě správně. Tento způsob se nazývá mock nebo mokování. Jde tedy o způsob, kdy spolupracující komponentu nasimulujeme tak, aby průběh testu nepoznal, že se nejedná o pravou třídu. Vytvoříme tím takovou třídu, která se chová podle skutečné třídy v aplikaci. Následující ukázka kódu představuje test na správné generování QR kódu.

```
@Test
public void generateQRCodeForAddressTest(){
    String path = System.getProperty("user.home");

    File fileFolder = new File(path+"/QRCode");
    folder.mkdir();

    Map⟨Object, String⟩ data = new HashMap⟨Object, String⟩();

    data.put("AddressMap", "AddresCity1");
    data.put("Address", "AddressCity1");

    QRCode.generateQRCodeForAddress(path, data);

    QRCodeReader reader = new QRCodeReader();
    BufferedImage image = ImageIO
    .read(new File(path+"/QRCode/AddressCity1.png"));
```

```
    int[] pixel = image.getRGB(0, 0,
    image.getWidth(), image.getHeight(), null, 0, image.getWidth());

    LuminanceSource source = new RGBLuminanceSource
    (image.getWidth, image.getHeight, pixel);
    BinaryBitmap bitmap = new BinaryBitmap(new HybridBinarizer(source));

     try{
         Result result = reader.decode(bitmap);
         Assert.assertEquals("AddressCity1", result.getText());
     }catch(Exception e){}
    }
```

Obrázek 6.1. Test na správné generování QR kódu

6.2 Integrační testy

Integrační testy se zaměřují na testování aplikace v rámci propojení jednotlivých modulů. Účelem těchto testů je ověření toho, že jednotlivé moduly v aplikaci předávají jednotlivé zprávy ve správný čas a dokáží spolu bez chyb spolupracovat. Typickým příkladem integračních testů je třída, která používá reálné připojení k databázi. Důvod proč rozlišujeme jednotkové a integrační testy je ten, že můžeme otestovat jednotlivé části aplikace, ale všechny moduly nemusí dohromady správně fungovat. Pro páci s databází byla vytvořena metoda, která obsahuje parametry potřebné po indetifikaci správné databáze a poslední parametr určuje akci.

```
@Test
   public void insertTest(){
      Map⟨String, Object⟩ data = this.data;
      Config.workWithDataInDatabase(path+"/TestFolder.mdb", "Address",
      "Address", data, null, null, 1);
      List⟨Row⟩ dataRow = Config.workWithDataInDatabase
     (path+"/TestFolder.mdb", "Address", "Address", data, "FirstName",
      null, 2);
     Assert.assertEquals("Male", dataRow.get(0).get('Gender'));
     Assert.assertEquals.("A", dataRow.get(0).get("FirstName"));
     Assert.assertEquals("A", dataRow.get(0).get("LastName"));
     Assert.assertEquals("B 1", dataRow.get(0).get("Address"));
   }
```

Obrázek 6.2. Test na správné vkládání dat do databáze.

Kapitola 7
Testování aplikace

Při vyvoření systému je obvyklé, že by se měla testovat na cílových uživatelích, aby se zjistilo, jakým způsobem na aplikaci reagují a jaké nedostatky v přehlednosti aplikace má. Je vhodné vybrat takou cílou skupinu, která s užíváním aplikace souvisí a abychom si byli jisti závažností a počtu chyb, je vhodné k testování přizvat 6-12 subjektů. K testování bakalářské práce jsem tedy přizval 6 subjektů.

7.1 Metoda

Heuristická evaluace [1]

Při testování heuristickou evaluací se projdou všechny kroky, které vedou k dosažení cíle a pak se zkoumá, jestli někde nenastal problém. Heuristická evaluace má vlastnosti, které má mít uživatelské rozhraní a jsou rozděleny do deseti heuristik. Tato metoda se používá pro obecný popis uživatelského rozhraní a případu užití, kdy není dán postup, jakým způsobem dosáhnout cíle.

- **Viditelnost systému** - systém informuje uživatele, co se v danou chvíli děje a vrací mu zpětnou vazbu v rozumném čase.
- **Spojitost mezi systémem a reálným světem** - systém mluví jazykem uživatele a řadí informace.
- **Uživatelská kontrola a svoboda** - systém poskytne východ z nechtěných stavů.
- **Konzistence a standardy** - systém dodržuje zvyklosti dané formy.
- **Prevence chyb** - systém eliminuje chyby a umožňuje potvrzení volby.
- **Rozpoznání, žadné vzpomínání** - systém má dané možnosti a volby jasné a uživatel si nemusí pamatovat žádné informace.
- **Flexibilita a efektivita použití** - systém poskytuje klávesové zkratky, zrychlení a je vhodný pro nováčka pokročilého uživatele.
- **Estetika a minimalistický design** - systém zobrazí pouze důležité informace.
- **Pomoc uživatelům rozpoznat, diagnostikovat a vyvarovat se chyb** - systém poskytuje chybové zprávy, přesné údaje a konstruktivní návrhy řešení.
- **Nápověda a dokumentace** - systém má minimální napovědu a dokumentaci, které jsou lehce k nalezení.

Při nálezu nějakého nedostatku se uvede, jaký typ heuristiky byl porušen a dodá se k němu i stupeň závažnosti.

- **0** - není problém týkající se použitelnosti
- **1** - velmi malý problém, který není potřeba opravit při nedostatku času.
- **2** - malý problém, který se opraví v posledných fázích vývoje softwaru
- **3** - velký problém, který je nutno opravit
- **4** - velmi velký problém, který je nutno opravit ještě před vydáním systému

[1] Text je převzat z přednášek z uvedeného předmětu `https://cent.felk.cvut.cz/courses/Y39TUR/`

7.2 Cílová skupina

Aplikace je navržena tak, aby s ní dokázal pracovat běžně pracující uživatel s počítačem. Danému subjekt stačí pouze znalost základní práce s počítačem a nemusí se učit žádné speciální požadavky.

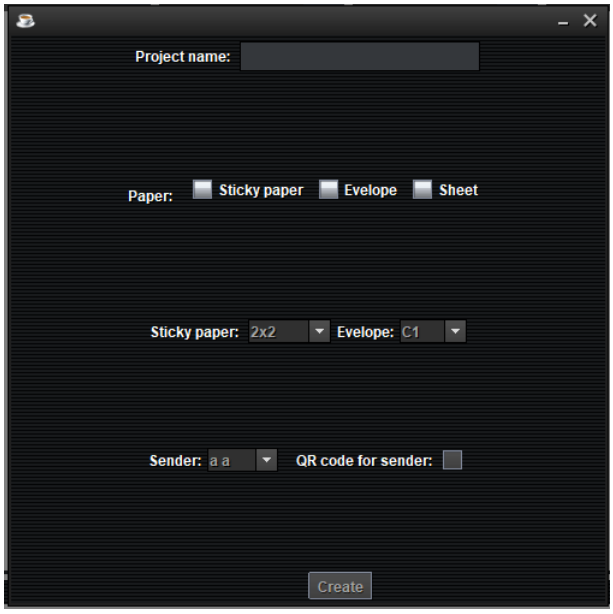
7.3 Testování

Každému z uživatelů byl poskytnutý ten samý stroj, na kterým aplikace běží se sadou úkolů, které mají být při testování splněny. Zadané úkoly zde uvedu ve formě případu užití.

1) Otevřít aplikaci
2) Pracovat s konkrétním papírem - na uživatelích bylo vyzkoušeno přídaní nového papíru a modifikace papíru
3) Založit projekt
4) Pracovat s adresou objektu - na uživatelích bylo vyzkoušeno napsání adresy a modifikace adresy
5) Zvolit jazyk
6) Uložit projekt
7) Otevřít projekt
8) Tisknout papír
9) Ukončit aplikaci

7.4 Nález

Název: Přidat odesílatele
Heuristika: 1
Stupeň: 4
Popis: Při prvním spuštění aplikace se do domovské složky uživatele nahraje databáze s předepsanými papíry a konfigurační soubor properties. Pokud si následně uživatel chce založit projekt, musí zadat také odesílatele. Aplikace samozřejmě neobsahuje předepsané odesílatele a tak je nutné před založením projektu zadat alespoň jednoho odesílatele. Uživatel obvykle klikne na položku Nový projekt a při vyplnění údajů pak zjistí, že nemůže vybrat odesílatele, pokud žádného nezadal do systému. Musí tedy okno nového projektu zavřít, otevřít okno s databází, založit odesílate a pak opět otevřít okno pro nový projekt, kde bohužel už nejsou vyplněny předchozí zvolené údaje 7.1.
Návrh řešení: Navrhoval bych umístit vedle rozbalovací položky s odesílateli nějaké tlačítko, které uživateli otevře okno pro vložení odesílatele do databáze. Pak se toto okno zavře a může dále pokračovat v zakládání projektu. Druhá možnost je vložit do tohoto okna výstrážné okno, které uživateli napoví, jakým způsobem má nejprve postupovat a znepřístupní všechny položky pro založení projektu před založením odesílatele do databáze. Tento nález byl velmi vážný a opravil jsem ho použitím druhé možnosti.



Obrázek 7.1. Nový projekt

Kapitola 8
Závěr

8.1 Zhodnocení

Smyslem mé bakalářské práce bylo především zkrácení času práce odpovědného pracovníka ve firmách, organizacích nebo státních institucích, který má na starost poštovní komunikaci se zákazníky atp. Vytvořil jsem jednoduchou intuitivní aplikaci, se kterou může pracovat kdokoli, aniž by ho bylo nutné speciálně školit. Aplikace od začátku vede uživatele krok po kroku k samotnému cíli. S touto aplikací se dříve nudná a zdlouhavá práce stává prací rychlou a efektivní a zároveň díky implementaci QR kódu zjednodušuje práci i samotnému doručovateli.

8.2 Návrh na budoucí vylepšení aplikace

- **Nápověda:** V aplikaci by bylo přínosné založit položku Nápověda, tak jak je to i v jiných aplikacích. Obsahovala by souhrné informace o tom, jakým způsobem používat aplikaci.
- **Změna hledání dat v mapách:** Změnu, kterou bych navrhl při zapisování adresu na papír, je přesunutí položky adresa na poslední místo. Tedy pořadí textových polí by bylo jméno, příjmení, město, poštovné směrovací číslo a adresa. Při zápisu města by se do kolonky poštovného směrovacího čísla zapsalo správné číslo a do kolonky adresy by se načetly všechny možné ulice, které se v daném městě nebo obci nachází. Zvýšila by se tím efektivnost a rychlost práce s aplikací.
- **Více archů:** Balíky se v dnešní době nemusí posílat jen přes Českou poštu, ale také přes různé přepravní společnosti. Do aplikace by se tedy mohli přidat různé formáty poštovních archů.
- **Čtení adres z různých souborů:** Aplikace by se dala vylepšit tím způsobem, že konkrétní zákazník by vypsal adresy do csv souboru v daném formátu a aplikace by mohla tento soubor přijmout a následně automaticky vložit všechny adresy na všechny papíry s příslušnými QR kódy.
- **Stát:** Při psaní adres se může stát, že budeme chtít poslat obálku do cizí země. V tomto případě by bylo užitečné přidat další textové pole pro zapsání státu.

Literatura

[1] SommerVille I.: *Softwarové inženýrství*, Computer Press, 2013

[2] Fowler M.: *Destilované UML*, Grada, 2009

[3] Shildt H.: *Mistrovství Java*, Computer Press, 2014

[4] Kiszka B.: *1001 tipů a triků pro jazyk Java*, Computer Press, 2009

[5] Pecinovský R.: *Úvod do objektové architektury pro mírně pokročilé*, Grada Publishing, 2014

[6] Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns*, Addison-Wesley, 2018

[7] Shirazi J.: *Java vyladování výkonu*, Grada Publishing, 2004

Příloha A
Přílohy

- **Zdrojové kódy:** Zdrojové kódy celého programu.
- **Diagramy UML:** Soubor EAP s navrhnutými diagramy.
- **Psaný projev:** Elektronická verze bakalářské práce.
- **Aplikace:** Soubor JAR se spustitelnou aplikací.
- **README:** Informacemi ke spustění programu.

Příloha B
Zkratky

ČVUT	České vysoké učení technické
GUI	Graphical user interface. Český název také grafické uživatelské rozhraní používané pro vzhled aplikací.
UML	Unified model language je grafický a návrhový jazyk pro popis softwarového systému.
JAVA	Programovací jazyk Java používaný pro programování aplikací.