**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačního inženýrství**


**Diplomová práce**

# Čisté objektově orientované programování

**Hana Křížová**

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Křížová Hana

Informatika

Název práce

**Čisté objektově orientované programování.**

Anglický název

**Pure object-based programming.**

---

**Cíle práce**

Cílem práce je vybrat několik jazyků a prostředí reprezentujících různé přístupy v objektové tvorbě softwaru, porovnat jejich vlastnosti a demonstrovat je na příkladě funkční aplikace části účetnictví.

**Metodika**

Seznámení se s různými prostředími objektových jazyků.
Analýza a návrh demonstrační aplikace.
Implementace aplikace podle zásad softwarového inženýrství a standardu UML.

**Harmonogram zpracování**

02-2013 / 10-2013 Sběr podkladů, seznámení se s prostředími.
11-2013 / 01-2014 Analýza, návrh a implementace aplikace.
02-2014 / 03-2014 Dokončení textu práce.

**Rozsah textové části**

60-80 stran

**Klíčová slova**

objektové jazyky, tvorba aplikací, informační systémy, softwarové inženýrství

**Doporučené zdroje informací**

Objective-C 2.0, Stehen G. Kochan , EAN:9788025126547 , nakladatelstvi computer press

Objektové programování, naučte se pravidla objektového myšlení, autor: Čada Ondřej, ISBN: 978-80-247-2745-5, nakladatelstvi Grada

OOP Objektově orientované programování bez předchozích znalostí, Autor:Jim Keogh, Mario Giannini, EAN:9788025109731 , nakladatelstvi: Computer press

Ruby - kompendium znalostí pro začátečníky i profesionály, Autor: Hal Fulton , Překlad: Jiří Koutný , ISBN 978-80-7413-018-2 , vydavatel: Zoner Press

**Vedoucí práce**

Merunka Vojtěch, doc. Ing., Ph.D.

**Termín odevzdání**

listopad 2014

Elektronicky schváleno dne 30.10.2013

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 5.12.2013

**Ing. Martin Pelikán, Ph.D.**

Děkan fakulty

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Čisté objektově orientované programování" jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 30.11.2014 

Poděkování

Ráda bych touto cestou poděkovala doc. Ing. Vojtěchovi Merunkovi, Ph.D. za poskytnutí cenných rad a trpělivost při vedení této diplomové práce.

# Čisté objektově orientované programování

---

# Pure object-based programming


**Souhrn**

Tato diplomová práce se zabývá čistě objektově orientovanými programovacími jazyky. V teoretické části jsou rozebírány programovací objektové jazyky Ruby, Python, Objective C, Javascript, framework Ruby on Rails pro tvorbu webových aplikací a dvě prostředí pro vývoj aplikací, a to NetBeans IDE s pluginem Ruby a Python a Eclipse IDE s pluginem Ruby a Python. V praktické části je rozebrán návrh, analýza a implementace konkrétní části aplikace účetnictví, a to zadání dodavatelských faktur, jejich zaúčtování, platby a návaznost na vykazování DPH. Tato část účetnictví je implementována v programovacích jazycích Ruby a Python ve vývojových prostředích NetBeans a Eclipse. Hlavním cílem této práce je tvorba aplikace účetnictví v různých programovacích jazycích a prostředích a jejich následné porovnání. Tohoto cíle je dosaženo navržením aplikace účetnictví ve vývojových prostředích NetBeans 7.4 a Eclipse, kde obě dvě vývojová prostředí zahrnují pluginy pro programovací jazyky Ruby a Python a závěrečným porovnáním vzniklých rozdílů u programovacích jazyků Ruby a Python a vývojových prostředích NetBeans IDE 7.4 a Eclipse IDE.



**Summary**

This thesis deals with pure object based programming langauges. In the theoretical part there are programming languages discussed such as Ruby, Python, Objective C, Javacsript, Framework Ruby on Rails for creating web application and two IDE Netbeans with plugins Ruby and Python and IDE Eclipse with the plugins Ruby and Python. In the practical part there are discussed a plan, analysis and implementation of the specific part of the application for the accountancy which means entering of invoices, their booking, payments and VAT. This part of the accountancy is implemented in the programming languages Ruby and Python in IDE NetBeans and Eclipse. The main goal of this thesis is a creation of the application for the accountancy in different programming languages and IDE and their comparison. This goal is achieved by designing the

accountancy application in IDE NetBeans and Eclipse, where both integrated development environment contain the plugins for the programming languages Ruby and Python and by final comparison of arised differences in programming languages Ruby and Python and IDE NetBeans and Eclipse.

**Klíčová slova:** objektové jazyky, tvorba aplikací, informační systémy, softwarové inženýrství

**Keywords**: object programming languages, creating of application, informatics systems, software engineering

# Obsah

# Seznam obrázků

# 1 Úvod

Tato diplomová práce je na téma Čisté objektově orientované programování. Objektově orientované programování má počátek v 60. letech minulého století a do dnešní doby jeho oblíbenost vzrostla.

V současnosti se vyskytují jak čistě objektově orientované programovací jazyky, ale také hybridní, tzv. smíšené programovací jazyky, které mají prvky objektově orientovaného programování. Mezi smíšené programovací jazyky patří i programovací jazyk Java. Jedním z čistě objektových jazyků je Objective C 2.0, který se těší velkému zájmu.

Tato diplomová práce se zabývá programovacími jazyky, které jsou čistě objektově orientované. Je zde zmíněn programovací jazyk Ruby, Objective C, Python, framework pro web Ruby on Rails a také, jak je objektový Javascript.

Mimo jiné jsou zde prezentována dvě vývojová prostředí, a to NetBeans IDE (verze 7.4) a Eclipse IDE, která mají doinstalovány pluginy pro Ruby a Python.

V praktické části bude názorně ukázán příklad s tvorbou aplikací pro účetnictví, kdy bude analyzován daný problém, také vytvořen návrh a následně vlastní implementance dané aplikace.

V závěru budou porovnány kódy, které budou vytvořeny ve vývojových prostředích výše zmíněných objektových jazycích Ruby a Python.

# 2 Cíl práce a metodika

## 2.1 Cíl práce

Cílem práce je vybrat několik objektových jazyků a prostředí reprezentujících různé přístupy v objektové tvorbě softwaru, porovnat jejich vlastnosti a demonstrovat je na příkladu funkční aplikace části účetnictví.

## 2.2 Metodika

Tohoto cíle je dosaženo seznámením se s programovacími objektovými jazyky Ruby, Python, Objective C, Javascript a seznámením se s vývojovými prostředími NetBeans IDE a Eclipse IDE.

Pro demonstraci je analyzována, navržena a implementována demonstrační aplikace z účetnictví.

V závěru této práce jsou rozebrány a porovnány rozdíly, vlastnosti objektových jazyků Ruby a Python, které vznikly při tvorbě aplikace.

Podkladem pro tuto práci byly odborné publikace v tištěné formě či elektronické formě.

# 3 Teoretická část

## 3.1 Objektové programování

*Objektové programování je založeno na volné síti vzájemně komunikujících objektů.(*Čada, 2009*)*[1] U objektově orientovaného programování se můžeme setkat se dvěmi různými koncepcemi, a to čisté objektově orientované programování a smíšené objektově orientované programování.

*Čisté objektově orientované programování je založeno pouze na objektech. Jeho praktická implementace na dnešních počítačích však není jednoduchá. Na rozdíl od předchozích softwarových paradigmat je totiž objektové paradigma na značně vyšší úrovni abstrakce, která neodpovídá tomu, jak počítače uvnitř pracují.* (Merunka, 2008)[2]

Smíšené objektově orientované programování *jsou založeny na imperativním programování a obvykle pouze částečně implementují vlastnosti objektového programování.(Wikipedie)*[3]

## 3.2 Základní prvky OOP:

Základními prvky objektově orientovaného programování jsou objekt, třída, kolekce, atribut, metoda, zpráva.

### 3.2.1 Objekt

*Na objekt lze nahlížet jako na stavební jednotku, která modeluje nějakou část reálného světa. V objektu jsou pohromadě uzavřena data i jejich vlastnosti a chování. Objekty jednoho systému jsou mezi sebou propojené různými vazbami a vzájemně na sebe působí.* (Merunka, 2008)[4]

1 ČADA, Ondřej. Objektové programování naučte se pravidla objektového myšlení. 1. vydání Praha: Grada Publishing, a.s. 2009 s.200 ISBN 978-80-247-2745-5, str.19

2 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.24

3 zdroj http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD

4 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.25

Co všechno může modelovat objekt? Objekt může modelovat, dá se říct, vše, co si představíme – např. studenta, nějaké zvíře, technické zařízení, proces nákupu atd.

Typy objektů:

- dynamické objekty
- automatické objekty
- statické objekty
- trvalé objekty

### 3.2.2 Abstraktní objekty a instance

O abstraktním objektu můžeme říci, že definuje základ pro skutečný objekt, který chceme popsat. Definuje jeho vlastnosti (atributy), ale neuvádí jejich konkrétní hodnoty. *Skutečný objekt se označuje jako instance abstraktního objektu.* (Keogh, Giannini, 2010)[5]

### 3.2.3 Třída

*Třída je šablona, která deklaruje atributy a definuje metody skutečných objektů.* (Keogh, Giannini, 2010)[6]

### 3.2.4 Kolekce objektů

Kolekci objektů si můžeme představit jako souhrn více objektů. Druhy kolekcí objektů jsou např. Set, Bag, List.

Set (Množina) *je kolekce, ve které prvky nemají žádné uspořádání. Přidává-li se do této kolekce prvek, který v kolekci již je, zůstává v této kolekci jen jednou. Tento druh kolekcí jako jediný přesně odpovídá matematickému pojetí množin.* (Merunka, 2008)[7]

5 KEOGH, Jim, GIANNINI, Mario. OOP bez předchozích znalostí. Průvodce pro samouky. dotisk 1. vydání Brno: Computer Press, a.s. 2010 s. 222 ISBN 80-251-0973-9, str. 15

6 KEOGH, Jim, GIANNINI, Mario. OOP bez předchozích znalostí. Průvodce pro samouky. dotisk 1. vydání Brno: Computer Press, a.s. 2010 s. 222 ISBN 80-251-0973-9, str 22

7 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s. 197 ISBN 978-80-87197-04-2, str. 30

Bag (Ranec) *je kolekce, ve které prvky také nemají žádné uspořádání. Přidává-li se ale do této kolekce prvek, který v kolekci již je, objeví se v této kolekci po takovém přidání více kopií tohoto prvku.* (Merunka, 2008)[8]

List (Seznam) *je kolekce, která se chová jako ranec a navíc zachovává vnitřní uspořádání prvků, které obsahuje.* (Merunka, 2008)[9]

### 3.2.5 Atribut

Atribut charakterizuje vlastnosti objektu, které objekt má. Např. v níže uvedeném jednoduchém příkladu máme objekt bankovní účet, který má atributy tři, a to SWIFT, BIC kódy a jméno vlastníka.


**Obrázek 1 - vlastní obrázek**

### 3.2.6 Metody

*Metody jsou funkce a procedury spojené s objektem, jejichž prostřednictvím je definováno jeho chování.*(Čada, 2009)[10]

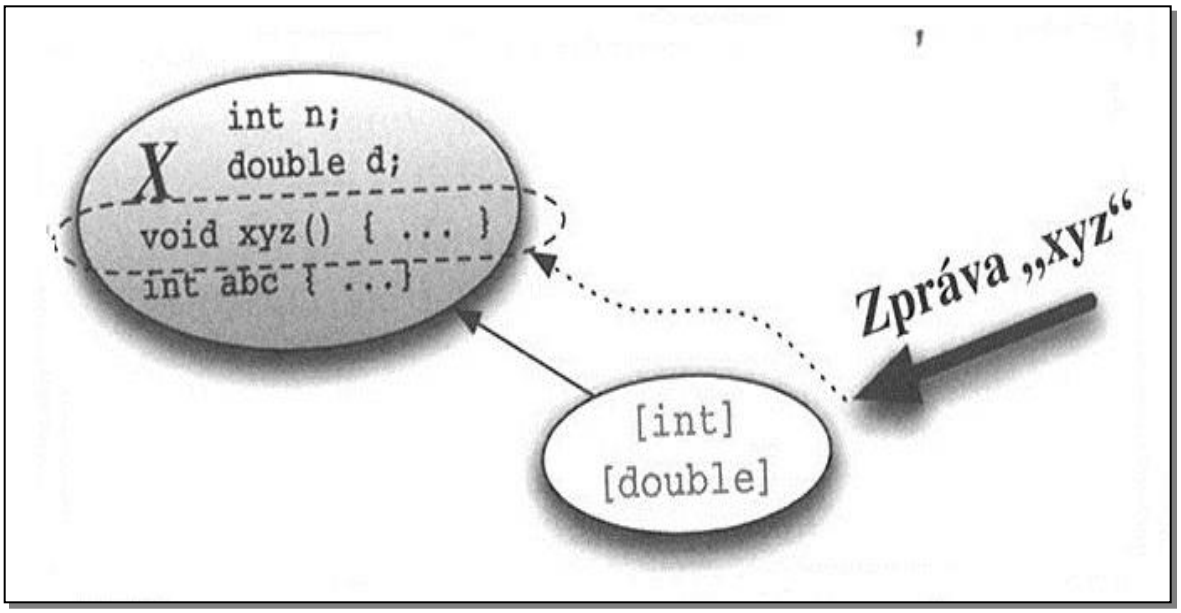
8 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.30

9 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.30

### 3.2.7 Zprávy

*Objekty mají tu vlastnost, že dokážou reagovat na požadavky, které jsou jim posílány. Požadavkům se v terminologii OOP říká zprávy. O objektech se potom hovoří, že jsou příjemci zpráv.*(Merunka, 2008)[11]

*Zpráva je obecný požadavek na nějakou činnost, který lze předat objektu; ten na ni obvykle reaguje vyvoláním vhodné metody, a to podle vlastního rozhodnutí.*(Čada, 2009)[12]




**Obrázek 2 - (Čada, 2009)**

## 3.3 Dědičnost, zapouzdření, polymorfismus

U objektového programování se setkáváme s dědičností, zapouzdřením a polymorfismem.

10 ČADA, Ondřej. Objektové programování naučte se pravidla objektového myšlení. 1. vydání Praha: Grada Publishing, a.s. 2009 s.200 ISBN 978-80-247-2745-5, str.28

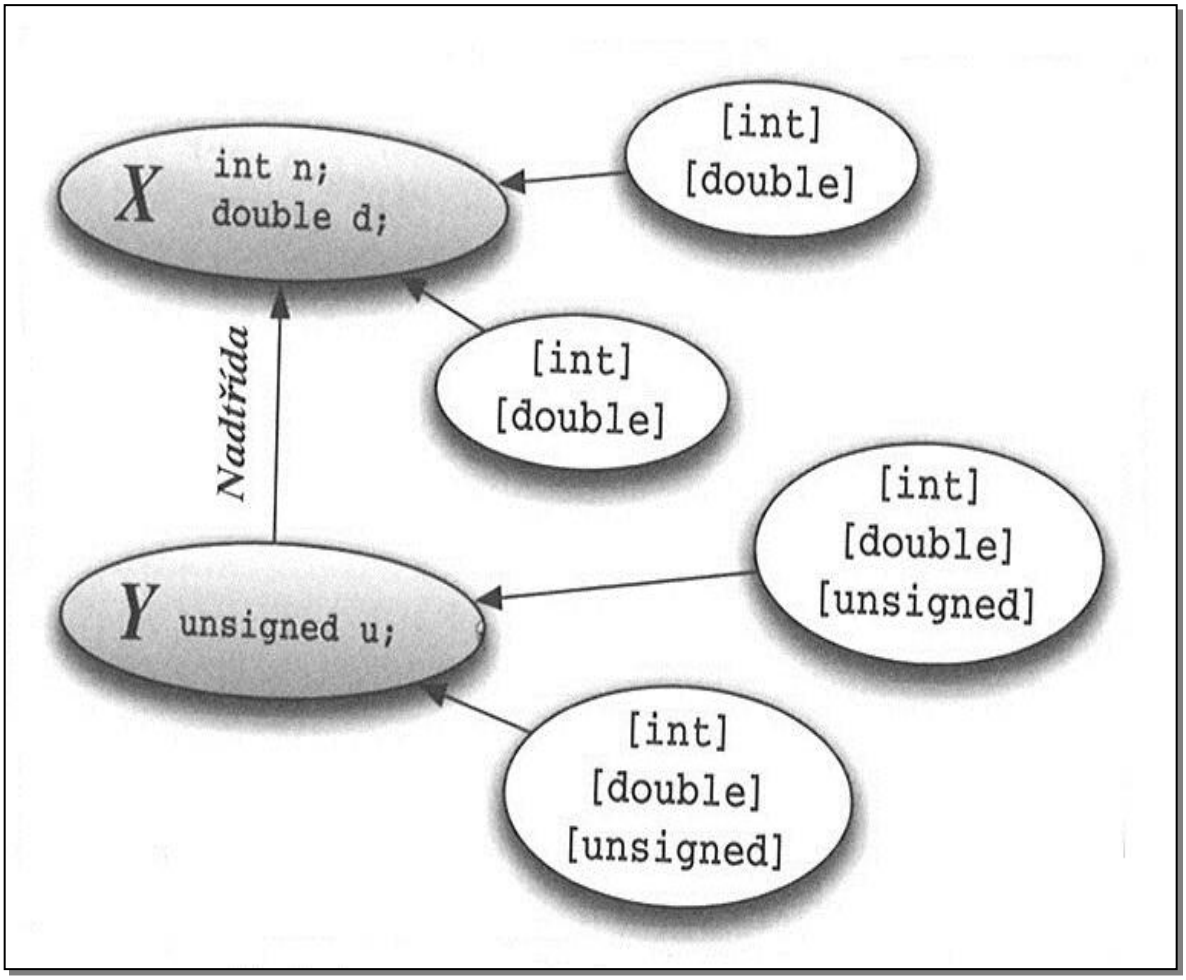
11 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.26

12 ČADA, Ondřej. Objektové programování naučte se pravidla objektového myšlení. 1. vydání Praha: Grada Publishing, a.s. 2009 s.200 ISBN 978-80-247-2745-5, str.28

### 3.3.1 Dědičnost

Dědičnost je používána jak v koncepci čisté objektově orientovaného programování, tak i ve smíšeném objektově orientovaném programování. Dědičnost je vlastnost objektů, resp. tříd. *Dědění mezi dvěma třídami znamená, že definice jedné třídy využívá definici druhé třídy. Třída, která má definici nejen pro sebe, ale i pro jiné třídy, se nazývá nadtřída. Třída, která takovou definici jiné třídy používá, se nazývá podtřída.*(Merunka, 2008)[13]

Dědění používáme v případě, kdy nechceme opakovat deklaraci atributů a metod u další podobné vytvořené třídy, protože bychom zbytečně opakovali již stanovené atributy a metody.




**Obrázek 3 - (Čada, 2009)**

13 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.34

### 3.3.2 Zapouzdření

Zapouzdření nám umožní, tzv. zneviditelnit objekt zvenčí. Objekty, které jsou kolem tohoto zapouzdřeného objektu, s ním mohou komunikovat, ale nemohou vidět jeho vnitřní uspořádání.

### 3.3.3 Polymorfismus

*Polymorfismus je vlastnost programovacího jazyka, objektově orientovaného programování (OOP), která umožňuje:*

- *jednomu objektu volat jednu metodu s různými parametry (ad-hoc polymorfismus)*
- *objektům odvozených z různých tříd volat tutéž metodu se stejným významem v kontextu jejich třídy, často pomocí rozhraní*
- *přetěžování operátorů neboli provedení rozdílné operace v závislosti na typu operandů (parametrický polymorfismus) (wikipedie)*[14]

*Polymorfismus znamená, že zpráva může vyvolávat různé operace, které se z pohledu toho, kdo zprávu poslal, jeví jako stejné, i když samy o sobě stejné nejsou. Jinými slovy to znamená, že pokud mají dva různé objekty shodné protokoly. Nemusí to ještě znamenat, že mají stejnou datovou strukturu a stejné metody.* (Merunka, 2008)[15]

## 3.4 Objektově orientované programovací jazyky

Mezi objektově orientované programovací jazyky se řadí např. C++, Java, C#, Ruby, Python, Objective C, Javascript, Smalltalk.

Tyto programovací jazyky se dělí na:

14 zdroj http://cs.wikipedia.org/wiki/Polymorfismus_(programov%C3%A1n%C3%AD)

15 MERUNKA, Vojtěch. Objektové modelování. 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2, str.28

- čistě objektově orientované jazyky, do kterých patří Ruby, Python, Objective C, Javascript.
- smíšené (hybridní) objektově orientované programovací jazyky, do kterých řadíme C++, Javu, C#, Visual Basic.

## 3.5 Čisté objektově orientované programovací jazyky a jejich prostředí

### 3.5.1 Ruby

Tvůrcem tohoto programovacího jazyka je Yukihiro "Matz" Matsumoto. V Ruby je vše objektem. Je také flexibilním jazykem, má různé implementace.

Implementace:

- MRI – Matz´s Ruby Interpreter
- CRuby
- JRuby
- Rubinius
- MacRuby
- Mruby
- IronRuby
- MagLev

Proměnné nemají definované typy, ale objekty mají typy.

*Ruby obsahuje všechny prvky, které se vztahují ke konceptu objektově orientovaného programování. Např. objekty se zapouzdřením a skrýváním dat, metody s polymorfismem a překrýváním, nebo třídy s hierarchií a dědičností. Ruby přidává omezenou podporu metatříd, singleton metod, modulů a mixinů.*(Fulton, 2009)[16]

16 FULTON, Hal. RUBY kompendium znalostí pro začátečníky i profesionály. 1. vydání Brno: ZONER software, s.r.o. 2009 s.769 ISBN 978-80-7413-018-2, str.59

Objekty v Ruby

*Všechna čísla, řetězce, pole, regulární výrazy a mnoho jiných entit jsou ve skutečnosti objekty. V Ruby je každý objekty instancí nějaké třídy. Třída obsahuje implementaci metod. Kromě zapouzdření vlastních atributů a operací má objekt v Ruby i svou identitu.* (Fulton, 2009)[17]

*Pro vytvoření objektu z existující třídy je obvykle použitga metoda new. Proměnné jsou používány k uchovávání refernecí na objekty.* (Fulton, 2009)[18]

Tvorba třídy

*Jazyk Ruby obsahuje mnoho vestavěných tříd. Dodatečné třídy mohou být definovány programově. Pro definici nové třídy se používá nnásledující konstrukce:*

```
class ClassName
#...
end
```

*Jméno třídy je globální konstanta, takže může z ačínat velkým písmenem. Definice třídy může obsahovat konstanty třídy, proměnné třídy, metody třídy, proměnné instance a metody instance. Data třídy jsou dostupná pro všechny objekty třídy, zatímco data instance jsou dostupná pouze pro jeden objekt. Jméno třídy je pouze konstanta, která je referencí na objekt typu Class (protože v Ruby je Class třída).* (Fulton, 2009)[19]

*Ruby je dynamický jazyk v tom smyslu, že objekty a třídy mohou být pozměněny v běhu programu. To znamená, že Ruby má schopnost vytvážen a vyhodnocovat kousky kódu během provádění existujícího (staticky naprogramovaného) programu.* (Fulton, 2009)[20]

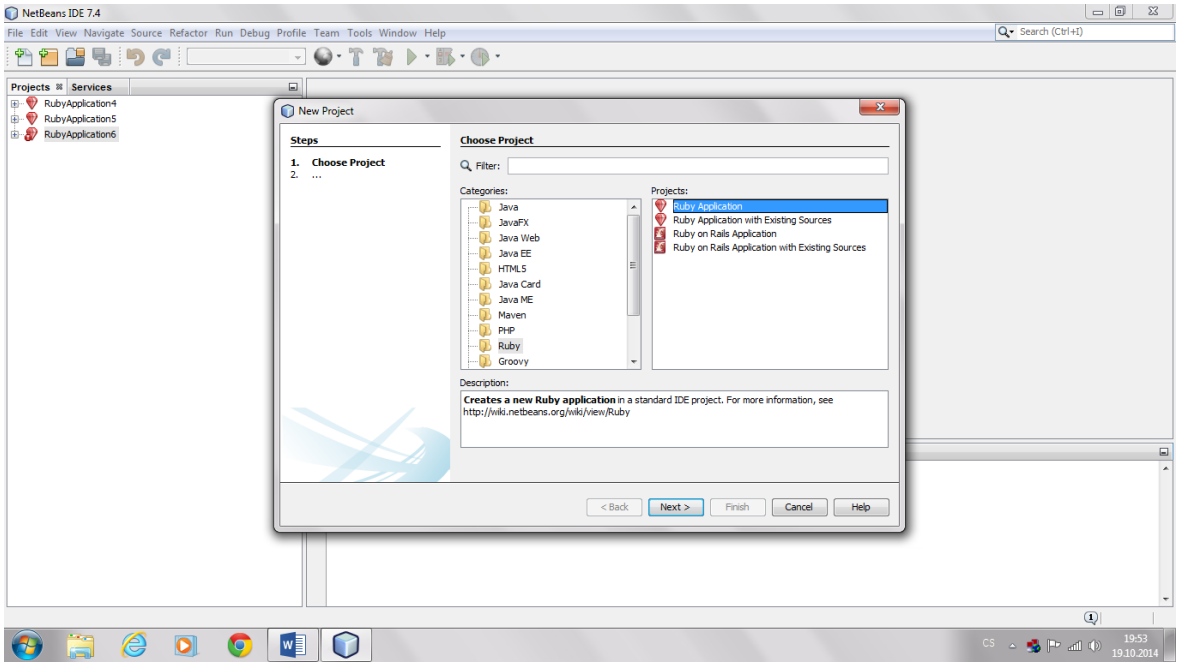
17 FULTON, Hal. RUBY kompendium znalostí pro začátečníky i profesionály. 1. vydání Brno: ZONER software, s.r.o. 2009 s.769 ISBN 978-80-7413-018-2, str.60
18 FULTON, Hal. RUBY kompendium znalostí pro začátečníky i profesionály. 1. vydání Brno: ZONER software, s.r.o. 2009 s.769 ISBN 978-80-7413-018-2, str.60
19 FULTON, Hal. RUBY kompendium znalostí pro začátečníky i profesionály. 1. vydání Brno: ZONER software, s.r.o. 2009 s.769 ISBN 978-80-7413-018-2, str.62,63
20 FULTON, Hal. RUBY kompendium znalostí pro začátečníky i profesionály. 1. vydání Brno: ZONER software, s.r.o. 2009 s.769 ISBN 978-80-7413-018-2, str.68

NETBEANS IDE 7.4 – plugin Ruby

Vývojové prostředí NetBeans IDE ve verzi 7.4. neobsahuje přímo při instalaci objektový jazyk Ruby, ale je možnost nainstalovat do tohoto vývojového prostředí plugin pro Ruby a Ruby on Rails.

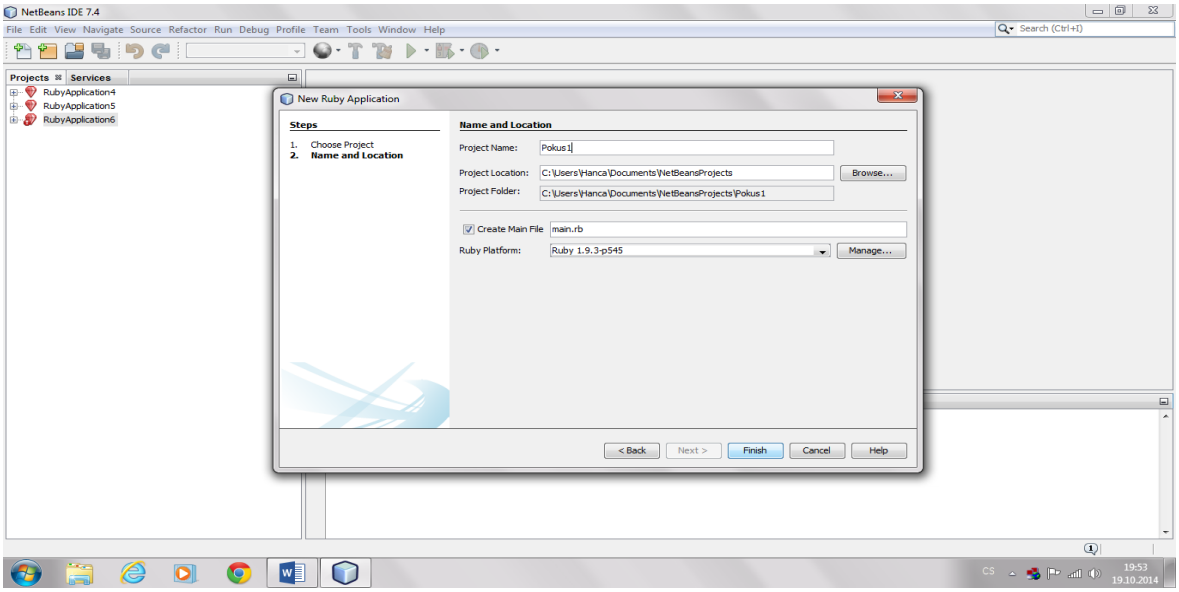
Uvedeme si zde ilustrační příklad tvorby projektu v tomto prostředí v programovacím jazyku Ruby.

Nejprve si zadáme nový projekt, kde se nám otevře nabídka programovacích jazyků, zde můžeme vidět, že se nám nabízí možnost vytvořit aplikaci v Ruby nebo v Ruby on Rails. Vybereme si tedy pouze Ruby. A přejdeme na další krok.



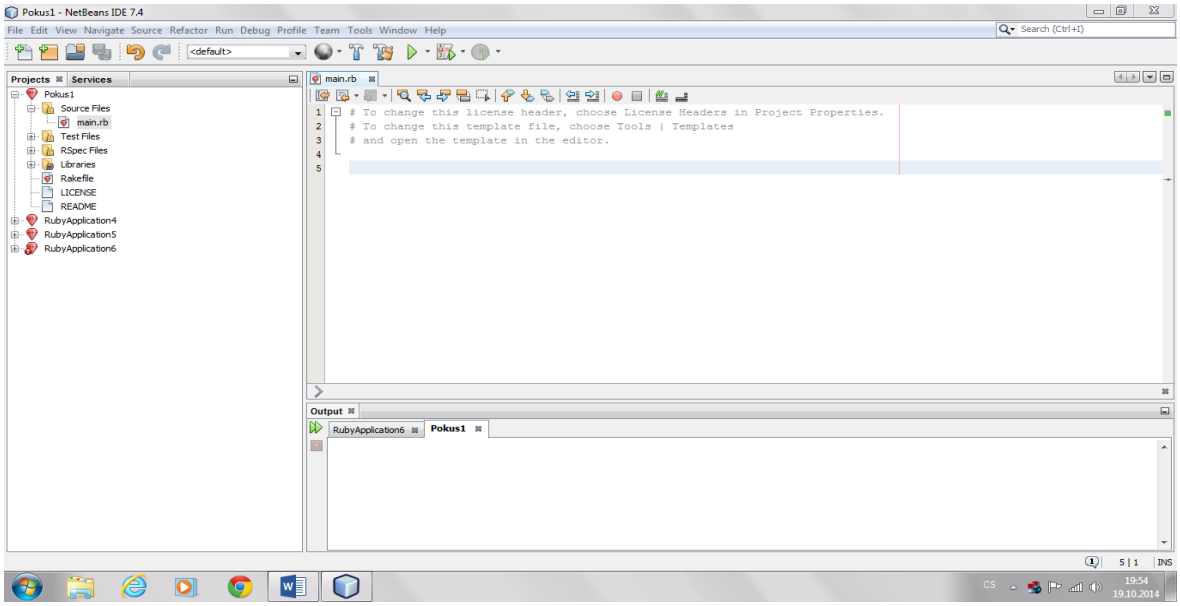

**Obrázek 4 - vlastní obrázek**

Otevře se nám nová Ruby aplikace, kde se zadá jméno projektu, místo uložení projektu a platforma Ruby a pro dokončení vytvoření se klikne na tlačítko Dokončit.




**Obrázek 5 - vlastní obrázek**

Po dokončení tohoto procesu se přímo otevře hlavní soubor (main.rb), kde se již může začít psát kód.




**Obrázek 6 - vlastní obrázek**

### 3.5.2 Ruby on Rails

*Ruby On Rails je framework pro vývoj webových aplikací napsaný v jazyce Ruby. Je navržen tak, aby usnadnil programování webových aplikací na základě obecných předpokladů vývoje pro web.*

*Filosofie Rails obsahuje několik vůdčích principů:*

- *DRY — „Don't repeat yourself" aneb „neopakujte se" — znamená, že psát stejný kód stále dokola je špatné a dobrý kód je „znovupoužitelný"*
- *Konvence má přednost před konfigurací — znamená, že Rails předpokládají co asi chcete udělat a jak to chcete udělat, místo aby vás nutily specifikovat každou drobnost v nekonečném množství konfiguračních souborů*
- *REST je nejlepší architektonický vzor pro webové aplikace — znamená, že uspořádat webovou aplikaci jako soubor „zdrojů"* (resources_) *a standardních HTTP „sloves"* (_verbs) *je nejsnazší a nejrychlejší způsob, jak modelovat entity a jejich vztahy*[21]

Architektura MVC

*Ruby on Rails obsahují ve svém jádru architekturu Model, View, Controller, obvykle označovanou jako MVC. Mezi výhody MVC patří:*

- *oddělení aplikační logiky od uživatelského rozhraní*
- *znovupoužitelnost kódu*
- *přehledná struktura kódu aplikace, která usnadňuje údržbu*

*Modely (models)*

*Model reprezentuje informace (data) ve vaší aplikaci a pravidla pro práci s nimi. V případě Rails jsou modely primárně využívány pro interakci s příslušnou tabulkou v*

[21] zpracováno dle zdroje: http://guides.rubyonrails.cz/

*databázi a pro ukládání pravidel této interakce. Ve většině případů odpovídá jedna tabulka v databázi jednomu modelu ve vaší aplikaci. Modely obsahují většinu aplikační logiky.*

*Pohledy (views)*

*Pohledy, neboli anglicky views reprezentují uživatelské rozhraní vaší aplikace. V Rails jsou viewsobvykle HTML soubory s vloženými částmi Ruby kódu, který provádí pouze úkony týkající se prezentace dat. Views mají na starosti poskytování dat webovému prohlížeči nebo jinému nástroji, který zasílá vaší aplikaci požadavky.*

*Kontrollery (controllers)*

*Kontrollery fungují jako „lepidlo" mezi modely a views. V Rails slouží kontrollery k zpracování požadavků které přichází z webového prohlížeče, získávání dat z modelů a k odesílání těchto dat doviews, kde budou zobrazeny.*

*Součásti Rails*

*Rails nejsou monolitický framework, ale jsou poskládány z mnoha dílčích součástí:*

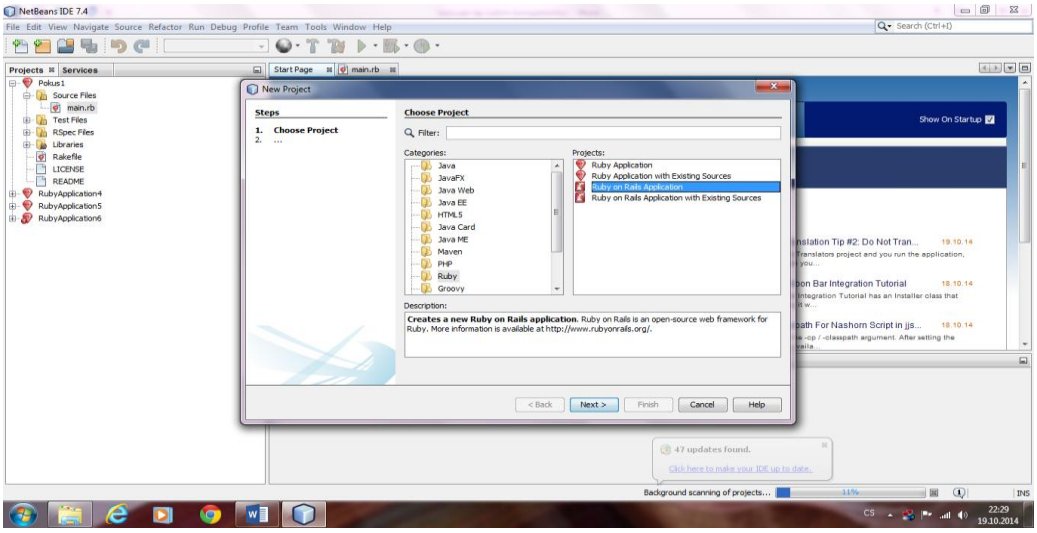
- *Action Pack*
  - *Action Controller*
  - *Action Dispatch*
  - *Action View*
- *Action Mailer*
- *Active Model*
- *Active Record*
- *Active Resource*
- *Active Support*
- *Railties*[22]

[22] zpracováno dle zdroje: http://guides.rubyonrails.cz/

### 3.5.3 Ilustrace tvorby projektu Ruby on Rails v NetBeans 7.4

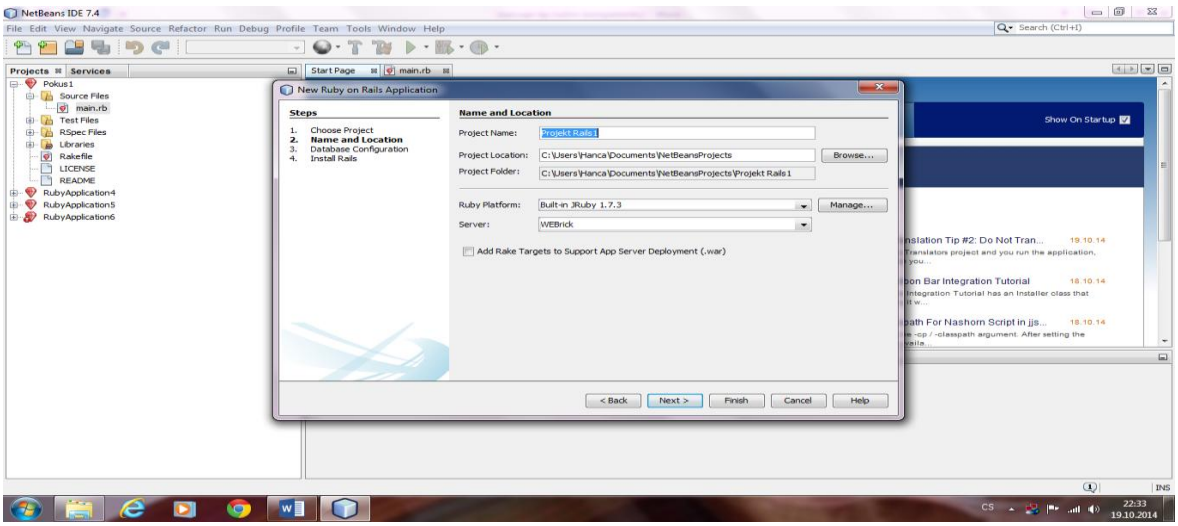
Zde je ilustrace příkladu na tvorbu projektu Ruby on Rails ve vývojovém prostředí NetBeans 7.4:

Otevře se nový projekt, vybere se Ruby on Rails aplikace a přejde se na další krok.



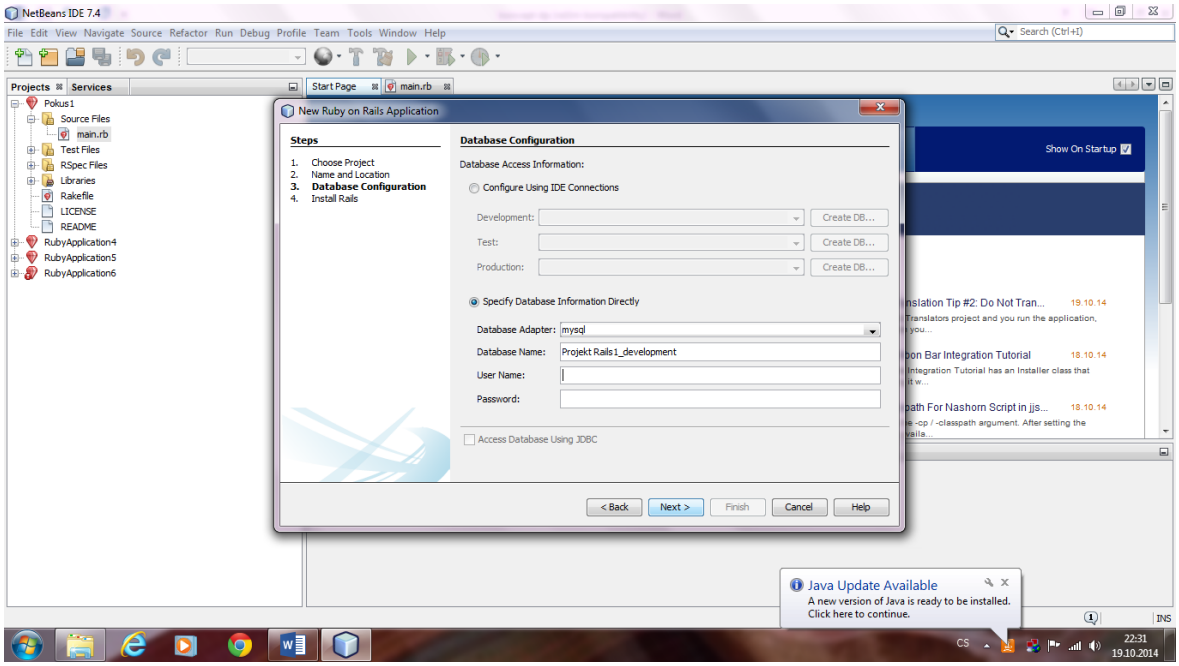

**Obrázek 7 - vlastní obrázek**

Zde se zadá název projektu, umístění projektu, plaforma Ruby a server a dá se dokončit tlačítkem Finish.



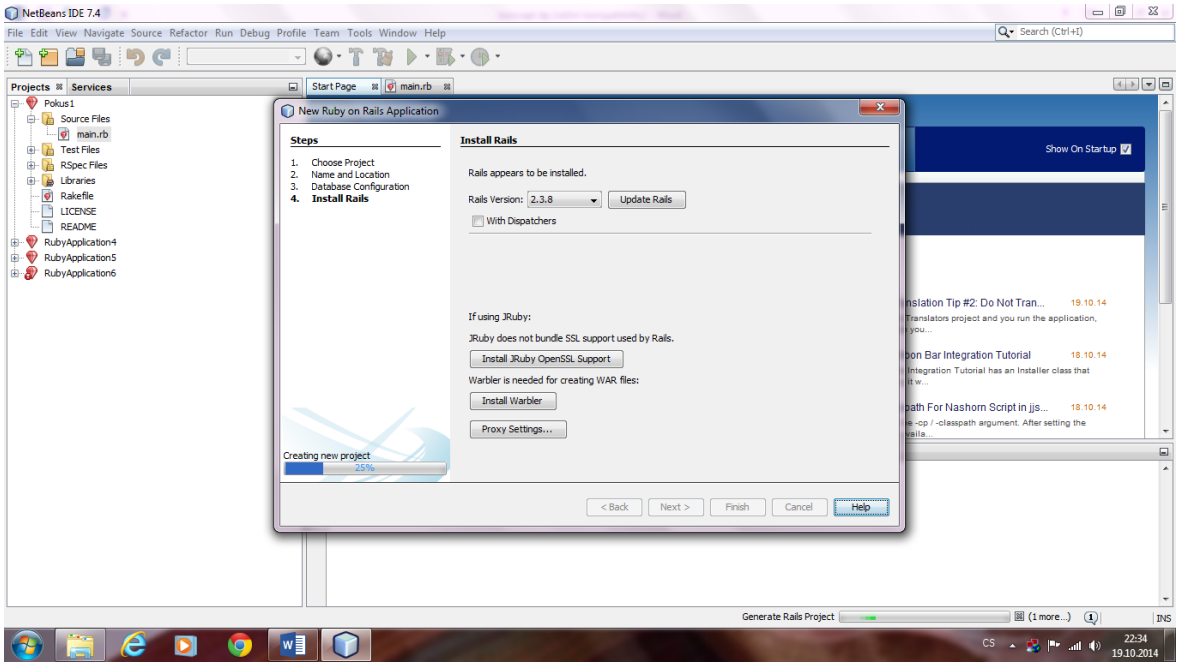

**Obrázek 8 - vlastní obrázek**

V dalším kroku se vybere databázový adaptér, zadá se jméno databáze, uživatelské jméno a heslo a přejde se k dalšímu kroku.




**Obrázek 9 - vlastní obrázek**

V následujícím kroku se nainstalují Rails.




**Obrázek 10 - vlastní obrázek**

### 3.5.4 Python

Tvůrcem tohoto programovacího jazyka je Guido van Rossum. *Python je multiplatformní jazyk. Je možné vytvářet programy napsané v Pythonu, které používají funkčnost specifickou pro určitou platformu. Jednou z opravdu silných stránek Pythonu je, že se dodává se skutečně kompletní standardní knihovnou, díky čemuž můžeme provádět třeba stahování souboru z Internetu, rozbalování zkomprimovaného archivního souboru nebo vytváření webového serveru jen pomocí jediného nebo několika málo řádků kódu. V jazyku Python lze programovat v procedurálním, objektově orientovaném a v menší míře též funkcionálním stylu, i když v jádru je Python objektově orientovaným jazykem.*(Summerfield, 2010)[23]

*Termíny třída, typ a datový typ používáme pro označení téhož pojmu. V jazyku Python můžeme vytvářet vlastní třídy, které jsou plně integrovány a které lze používat stejně jako vestavěné datové typy. Termín objekt a případně také instance, používáme pro označení instance určité třídy. Objekty mají obvykle atributy, přičemž metody jsou volatelné atributy a ostatní atributy jsou data. Datové atributy se běžně implementují jako proměnné instance, což jsou proměnné, které existují samostatně v každém objektu daného typu.* (Summerfield, 2010)[24]

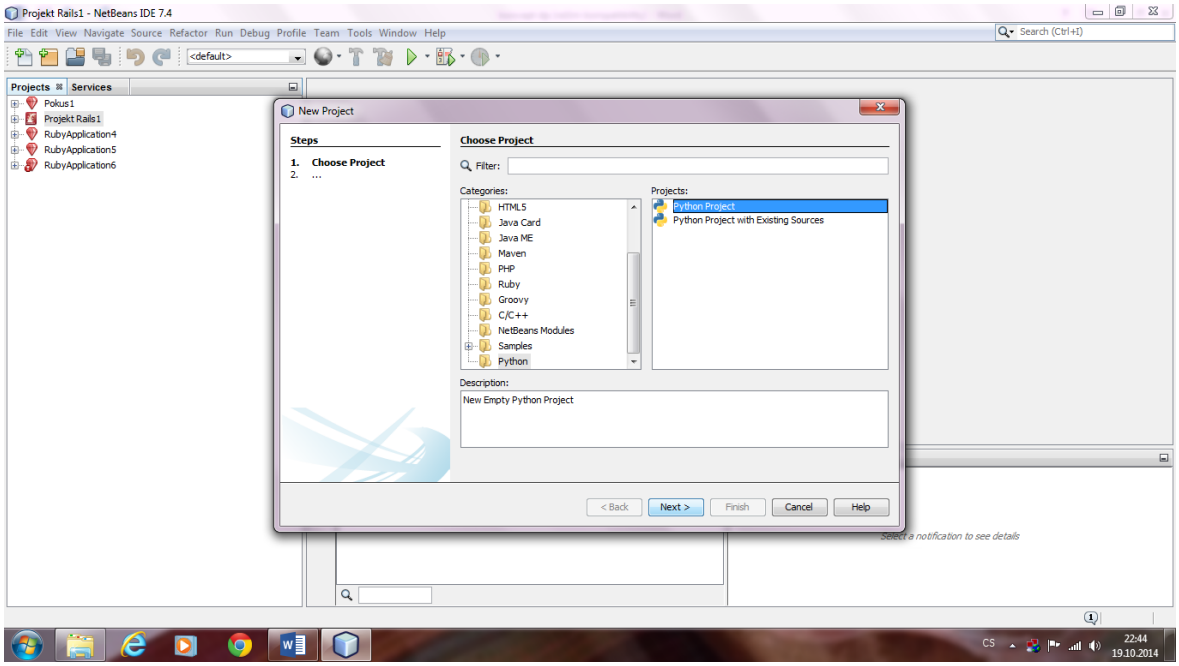
Ilustrace tvorby projektu v programovacím jazyku Python v NetBeans 7.4. Jak již bylo uvedeno u výše uvedeného Ruby, NetBeans ve verzi 7.4 nenabízí přímou instalaci s programovacím jazykem Ruby, ale také i Pythonu. Toto je řešeno doinstalováním pluginu Python.

Níže bude opět uveden ilustrační příklad tvorby nového projektu pro programovací jazyk Python ve vývojovém prostředí NetBeans 7.4.

Opět se otevře nový projekt, z nabídky programů se vybere Python, otevře se nabídka pod tímto objektovým jazykem a vybere se Python projekt a následně se přikročí k dalšímu kroku. Na níže uvedeném obrázku č. 11 je tento proces ukázán.

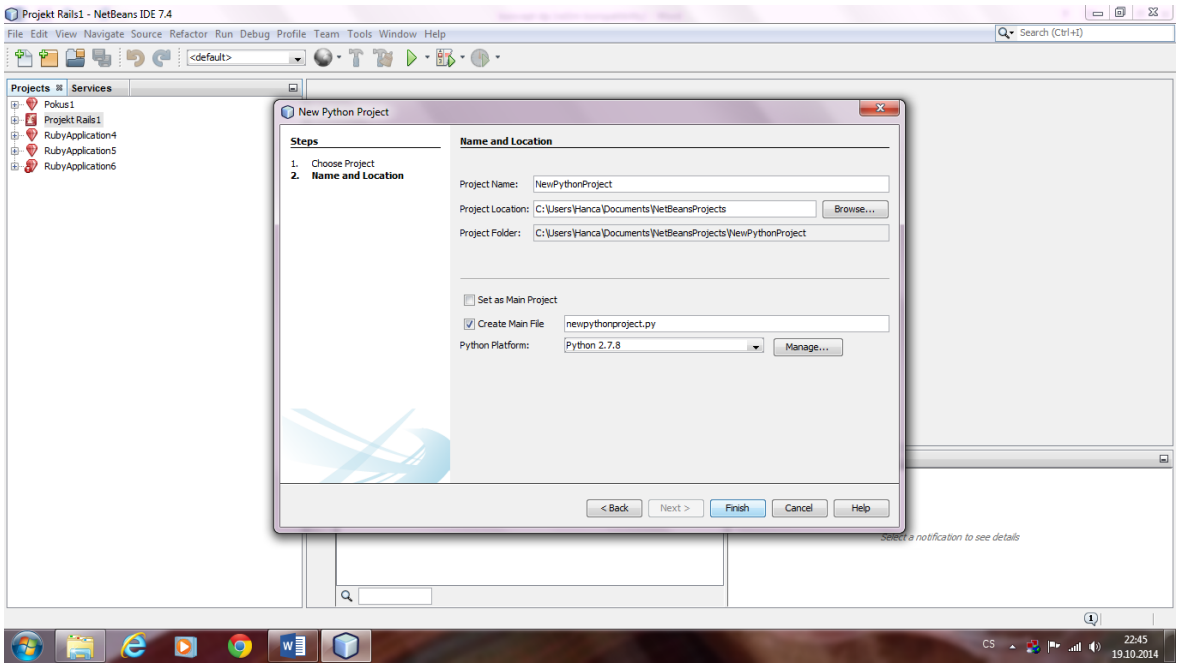
23 SUMMERFIELD, Mark. Python 3 Výukový kurz. 1. vydání Brno: Computer Press, a.s. 2010 s.584 ISBN 978-80-251-2737-7, str.13

24 SUMMERFIELD, Mark. Python 3 Výukový kurz. 1. vydání Brno: Computer Press, a.s. 2010 s.584 ISBN 978-80-251-2737-7, str.231-232



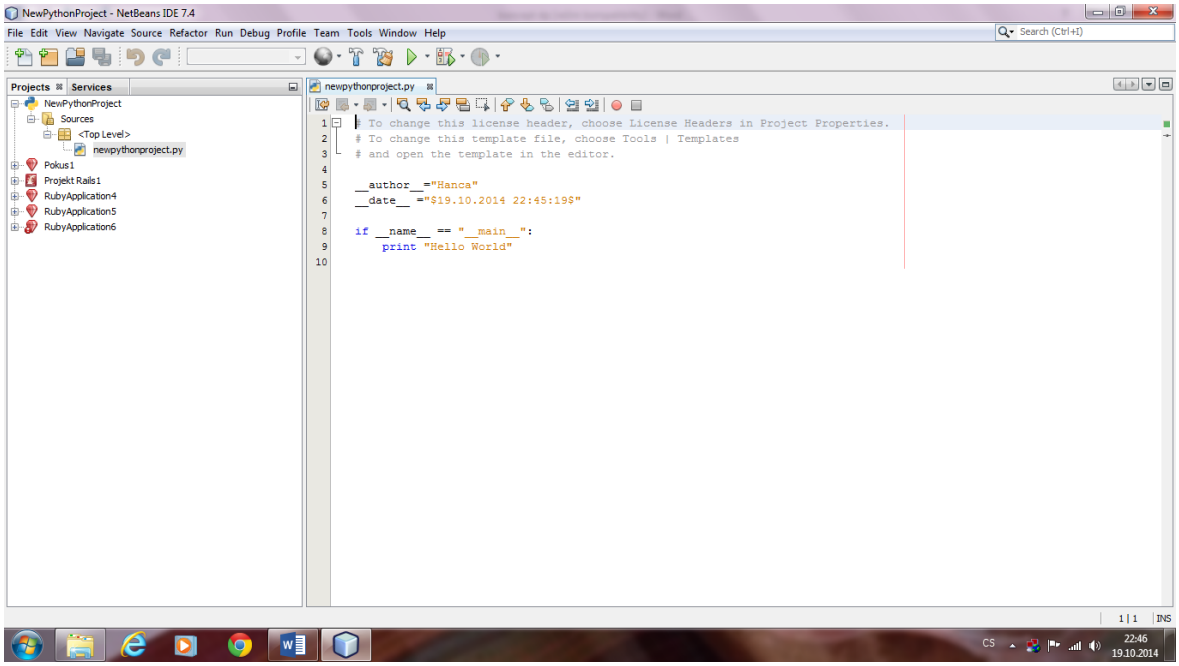

**Obrázek 11 - vlastní obrázek**

Zde se pojmenuje projekt, zvolí se umístění projektu, případě je možné nastavit, že tento projekt bude hlavní. Je také možné zvolit vytvoření hlavního souboru. V závěru se vybere platforma Pythonu. Po tomto je možné projekt uložit. Názorná grafická ukázka je na obrázku č. 12.




**Obrázek 12 - vlastní obrázek**

Po dokončení vytvoření projektu se již může začít s programováním.




**Obrázek 13 - vlastní obrázek**

### 3.5.5 Objective C

Tvůrci Objective C jsou Brad Cox a Tom Love. Poprvé se objevil v roce 1983. Objective C byl ovlivněn programovacími jazyky C a Smalltalk. Naopak Objective C ovlivnil programovací jazyky jako jsou: Java, Nu, Objective-J, TOM.

Současná verze Objective C je 2.0, kdy ji Apple v roce 2006 uvedl na Mezinárodní vývojové konferenci.

*Objective C je primární programovací jazyk pro tvorbu softwaru pro OS X a iOS. Objective C dědí syntaxi, primitivní typy jazyka C a přidává syntaxi pro definování tříd a metod. Objective C užívá protokoly k definování skupiny zahrnutých metod jako metody na objektu.*

*Syntaxe Objective C deklaruje třídu tímto způsobem:*

```
@interface SimpleClass : NSObject


@end
```

**Obrázek 14 - zdroj https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html**

*Tento příklad deklaruje třídu pojmenovanou SimpleClass která dědí from NSObject.*

*XCode je IDE pro tvorbu OS X a iOS softwaru. V Objective C je třída sama jako objekt s neprůhledným typem pojmenovaným Claas. Třídy nemohou mít vlastnosti definované užitím deklarované syntaxe ukázané dřív pro instance, ale moho obdržet zprávy.* [25]

### 3.5.6 Javascript

Javascript je dynamický, objektový skriptovací jazyk, který podporuje prototyp založený na objektové konstrukci.

*Javascript funguje jako procedurální i jako objektově orientovaný jazyk. Objekty jsou tvořeny v Javascriptu připojením metod a vlastností k dalším prázdným objektům v čase, kdy naopak k definici třídy běžně v kompilovaných jazycích jako je C++ a Java. Jakmile byl objekt vytvořen, může být použít jako plán nebo prototyp pro tvorbu podobných objektů.*

*Javascript je navržen na jednoduchém objektově orientovaném paradigmatu. Jako objekt je kolekce vlastností a vlastnost je asociací mezi jménem a hodnotou. Hodnota vlastnosti může být funkcí, v takovém případě je vlastnost známá jako metoda. Navíc k objektům, které jsou předdefinované v prohlížeči lze definovat vlastní objekty.* [26]

25 Volně přeloženo z https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html, originál textu v anglickém jazyce je přiložen v příloze A této diplomové práce

26 volně přeloženo z https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, originál textu v anglickém jazyce je přiložen v příloze A této diplomové práce

## 3.6 Vývojová prostředí NetBeans a Eclipse

### 3.6.1 NetBeans IDE

NetBeans IDE je vývojové prostředí pro různé programovací jazyky a aplikace. V současné době (tj. listopadu 201) je nejaktuálnější verze 8.0.1.

*NetBeans IDE umožní rychle a jednoduše rozvíjet Java desktop, mobilní a webové aplikace jako s HTML5 aplikace s HTML, CSS a Javascriptem. Toto IDE také poskytuje velkou nabídku nástrojů pro PHP a C/C++ vývojáře. Je to volné a open source vývojové prostředí.*

*Toto vývojové prostředí je víc než jen textový aditor. NetBeans editor odsazuje řádky, spojuje slova a závorky a označuje zdrojový kód syntakticky a sémanticky. Také poskytuje šablony, kódové nápovědy a nástroje pro refactoring.*

*Editor podporuje různé jazyky od Javy, C/C++, XML a HTML, k PHP, Groovy, Java-doc, Javascript a JSP. Protože editor je rozšiřitelný, je možné přidat plugin pro podporu dalších jazyků.*

*NetBeans IDE může být instalován na všechny operační systémy, které podporují Javu, od Windows po Linux a Max OS X systémy. NetBeans IDE je modulární vývojový nástroj pro širokou oblast vývojových aplikačních technologií. Základní IDE zahrnuje pokročilý multijazykový editor, debugger a profiler jako nástroje pro ovládání verzování a spolupráci.*

*NetBeans IDE dává kostru aplikací ve formě projektových šablon pro všechny technologie, které podporuje. Navíc poskytuje vzorové aplikace, některé z nich mohou být přetvořeny pomocí dostupného tutoriálu na NetBeans.org. Ide poskytuje projektové šablony a vzorové projekty, které pomohou s vytvořením Java SE apolikací, Java EE aplikací, Java ME aplikací, HTML5 aplikací, PHP aplikací a C/C++ aplikací.* [27]

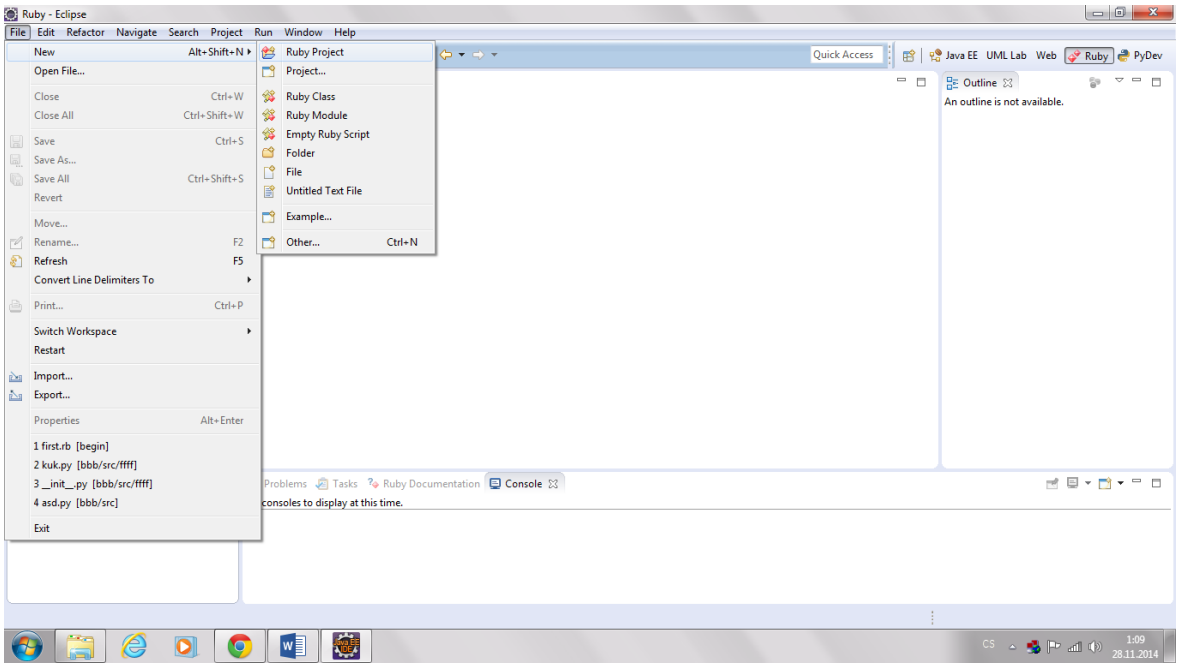
27 volně přeloženo z https://netbeans.org/features/index.html, originál textu v anglickém jazyce je přiložen v příloze A této diplomové práce

### 3.6.2 Eclipse

*Eclipse je známý pro Javu IDE, ale C/C++ IDE a PHP IDE jsou také velmi žádané. Lze snadno kombinovat podporu jazyků.*[28]

Zde je ilustrace tvorby aplikace ve vývojovém prostředí Eclipse v programovacím jazyku Ruby.

V pravé části se vybere programovací jazyk, v tomto případě Ruby, otevře se nový projekt přes Soubor/Nový/Ruby projekt

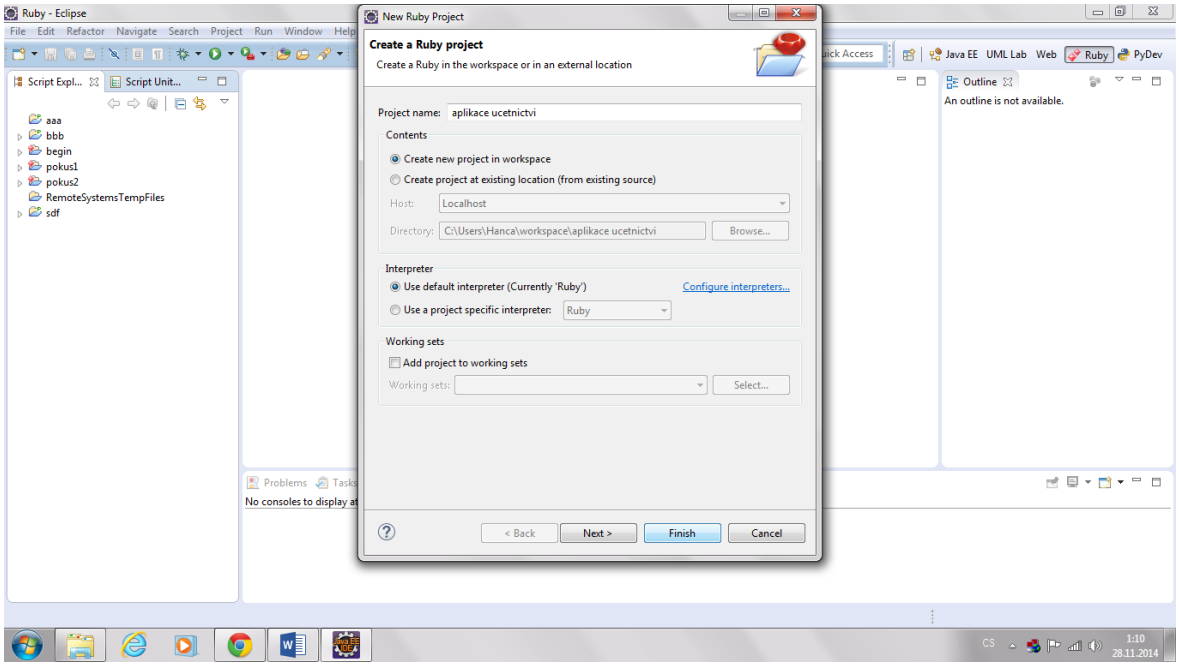



**Obrázek 15 – vlastní obrázek**

Po zadání Nového projektu se objeví níže uvedené nabídkové okno, kde se zadá název projektu, vybere se interpreter a dokončením se otevře daný projekt.

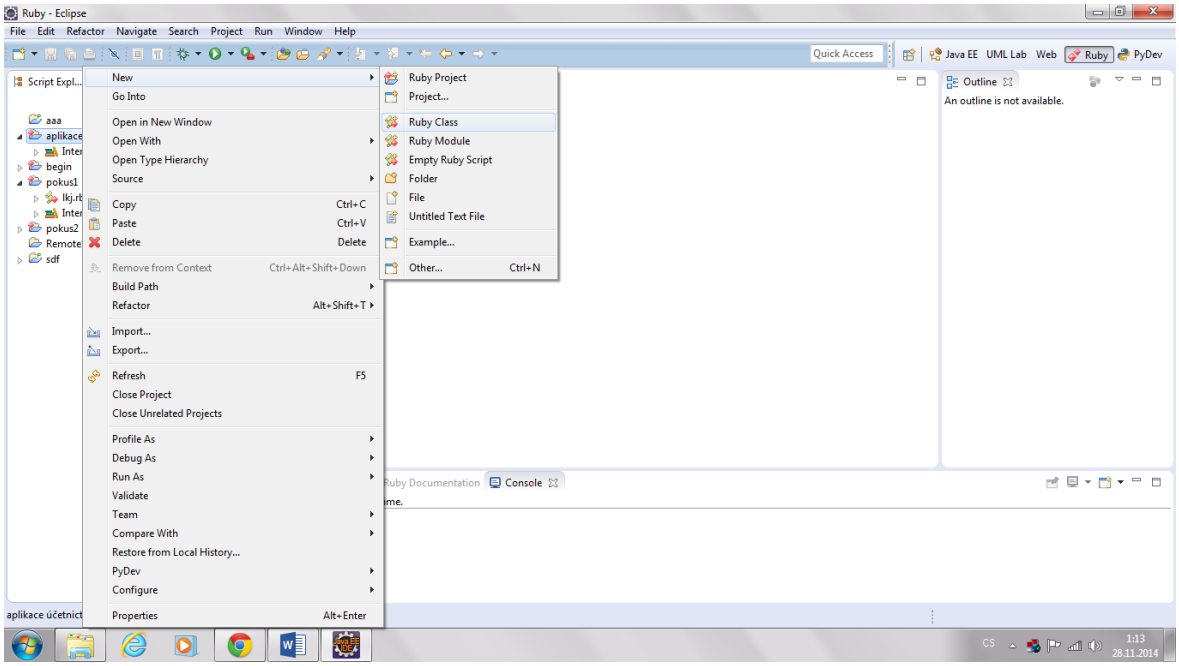
Z důvodu velikosti obrázku je obrázek č. 16 na další stránce.

28 volně přeloženo z https://eclipse.org/org/, originál textu v anglickém jazyce je přiložen v příloze A této diplomové práce



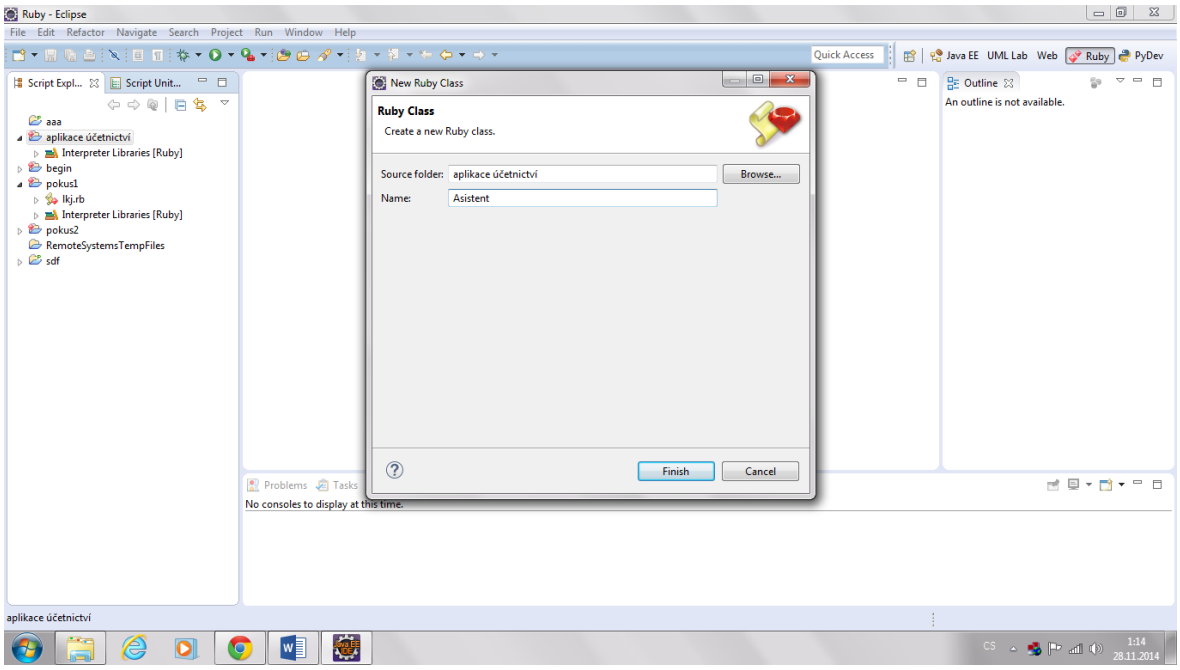

**Obrázek 16 - vlastní obrázek**

Podobným způsobem se mohou vytvářet v daném projektu třídy. Níže na obrázku je ilustrace tvorby třídy.



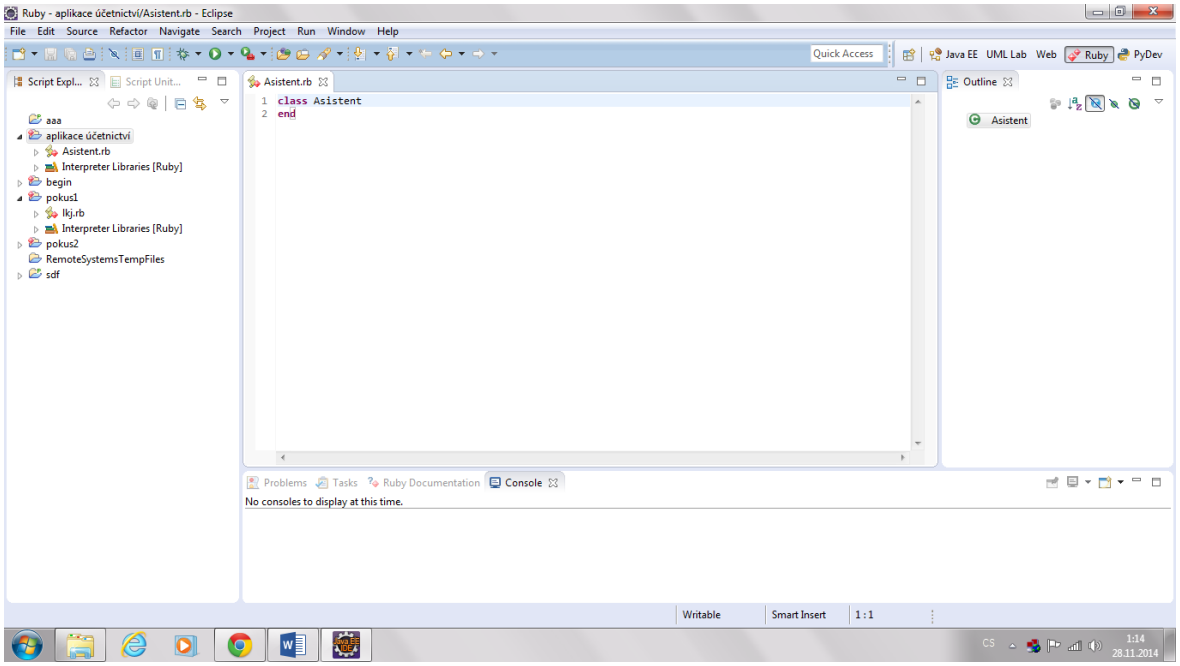

**Obrázek 17 - vlastní obrázek**

Po výběru „Ruby class“ se otevře nabídkové okno, kde se zadá název třídy a dá se dokončit. Ilustrace viz níže na obrázku.




**Obrázek 18 - vlastní obrázek**

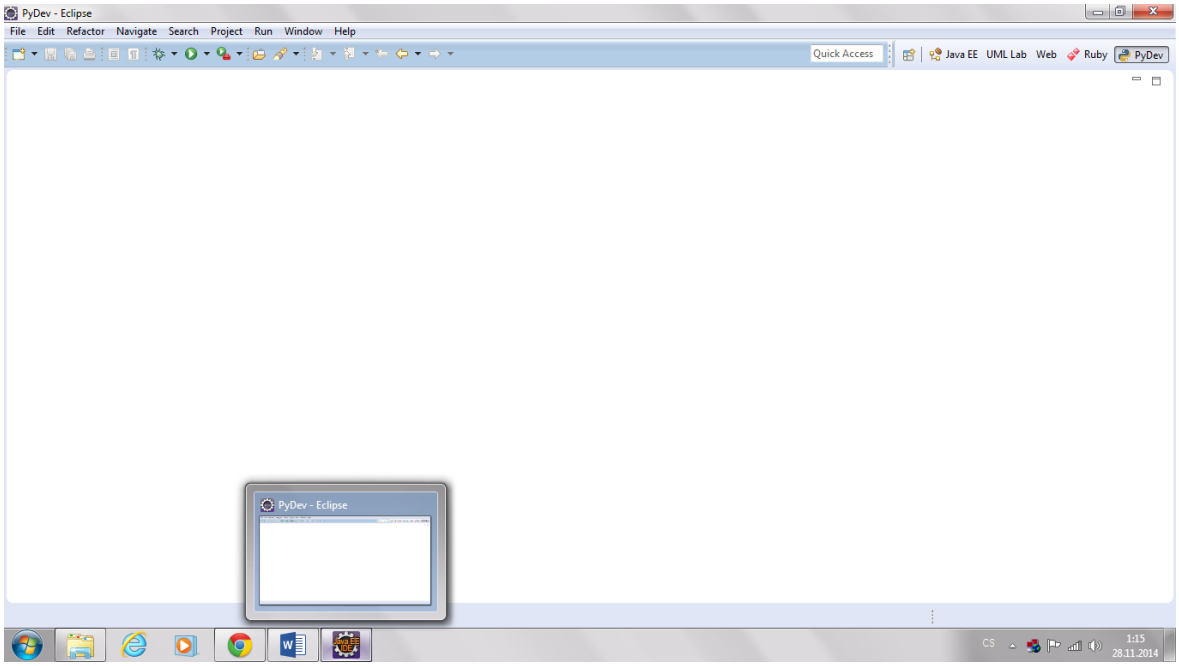
Po dokončení se otevře zdrojový kód třídy, který se může dále upravovat dle potřeb dané aplikace.




**Obrázek 19 - vlastní obrázek**

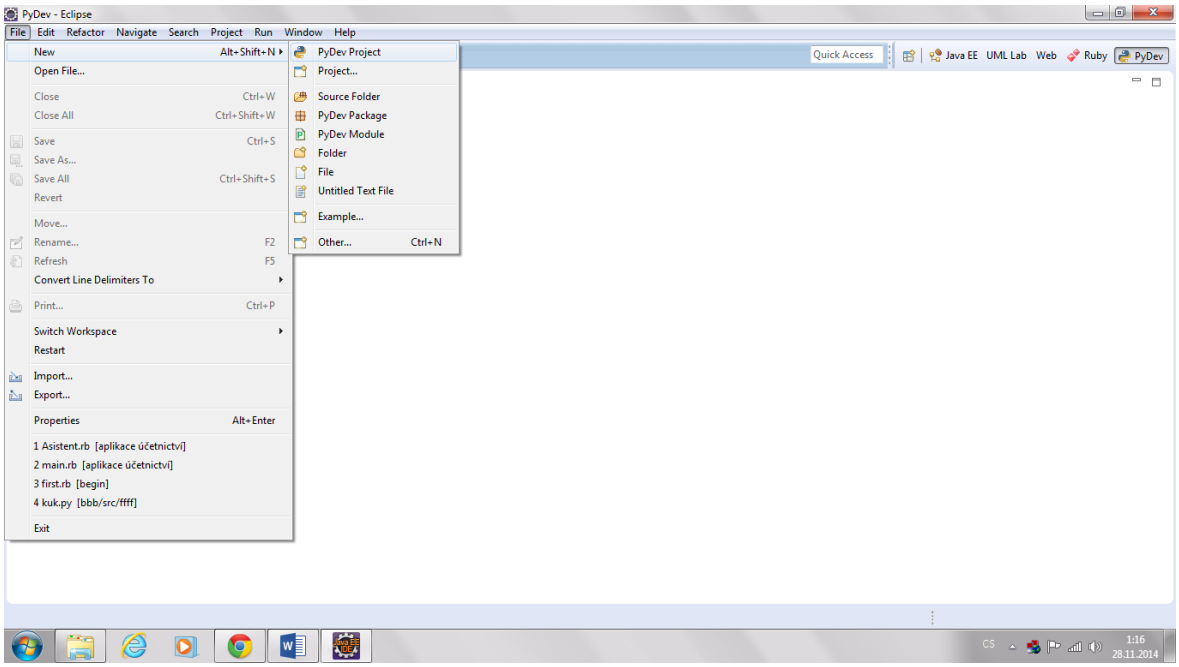
Zde je ilustrace tvorby aplikace ve vývojovém prostředí Eclipse v programovacím jazyku Python.

Opět se vybere programovací jazyk, a to přes platformu PyDev.



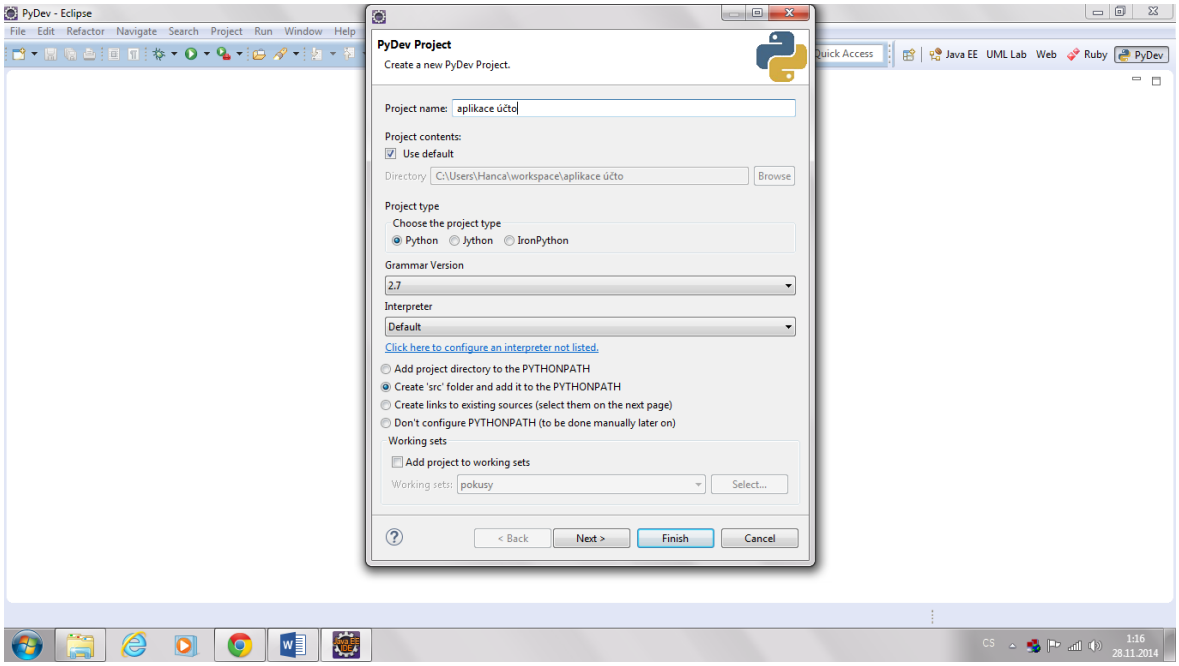

**Obrázek 20 - vlastní obrázek**

Se zadáním nového projektu je to podobné jako v předchozí ilustraci tvorby aplikace v programovacím jazyku Ruby.




**Obrázek 21 - vlastní obrázek**

Otevře se nabídkové okno, kde se pojmenuje daný projekt, vybere se typ Pythonu, verze, interpreter a dá se dokončit. Ilustrace viz níže.




**Obrázek 22 - vlastní obrázek**

# 4 Praktická část

V praktické části této diplomové práce se budeme zabývat analýzou, návrhem a implementací aplikace účetnictví pro firmu, zabývající se výrobou elektronických výrobků.

Od zákazníka jsme obdrželi požadavek vytvořit aplikaci účetnictví, která bude zahrnovat tyto oblasti:

a) tvorba účtového rozvrhu s možností založení analytických účtů
b) příjem a zadání dodavatelských faktur
c) zaúčtování těchto faktur
d) platby dodavatelských faktur
e) fakturaci zákazníkům
f) zaúčtování vystavených faktur

g) zaúčtování skladových zásob
h) pokladna
i) banka
j) komplexní evidence a výkaznictví DPH včetně elektronických výstupů pro FÚ
k) tisk faktur, pokladních přímových a výdajových dokladů
l) evidence, odpisy a účtování o majetku
m) zaúčtování mezd
n) možnost účtování v Kč a v jiných měnách
o) volba režimu účtování – zda-li v kalendářním roce nebo po sobě jdoucích 12ti měsíců

Z důvodu rozsahu diplomové práce se budeme zabývat těmito oblastmi:

a) příjem a zadání dodavatelských faktur
b) zaúčtování těchto faktur
c) platby těchto faktur
d) zpracování DPH

## 4.1 Analýza zadaného problému

Pro správné provedení analýzy je třeba mít podrobné odpovědi na níže uvedené otázky, z toho důvodu, aby navrhované diagramy odpovídaly tomu, co si přeje zákazník.

**Navrhované otázky:**

- **Kde a kým bude aplikace používána?**
  Aplikace bude používána asistentem účetního oddělení, účetní a hlavní účetní. Asistent bude zadávat faktury a dodavatele do informačního systému. Účetní bude kontrolovat tato zadání faktur a následně bude faktury zaúčtovávat a provádět jejich platby. Hlavní účetní provede kontrolu zaúčtování a následně si připraví podklady pro zpracování DPH.

- **Jak bude aplikace plnit své poslání, cíl?**

Umožní zadání všech dat, potřebných pro účetnictví a výkaznictví DPH.

- **Jsou specifikovány oblasti přístupu účastníků do aplikace?**
  Ano, asistent má na starosti správné zadání faktury do aplikace a zadání nového dodavatele.

- **Jaká jsou omezení?**
  Faktura musí být před zadáním do aplikace vždy schválena. Přístup do informačního systému bude vždy povolen pouze registrovaným uživatelům, tj. po zadání loginu a hesla.

- **Jaké jsou požadavky pro zadání dodavatelských faktur?**
  Faktura musí splňovat všechny náležitosti daňového dokladu.

Po získání potřebných odpovědí na zadané otázky je možné přistoupit k tvorbě Use Case diagramů. Pokud by tvůrce aplikace neměl dostatečné informace od zákazníka, nemusela by vytvořená aplikace plně vyhovovat požadavkům zákazníka. Během analýzy musí být tvůrce aplikace v kontaktu se zákazníkem a vyžadovat od něj doplňující informace.

### 4.1.1 Use Case Diagram pro aplikaci účetnictví

Vzhledem k tomu, že zde budou celkem tři účastníci, nejdříve bude Use Case diagram a popis Use Case diagramu zhotoven celkem třikrát pro každého účastníka zvlášť.

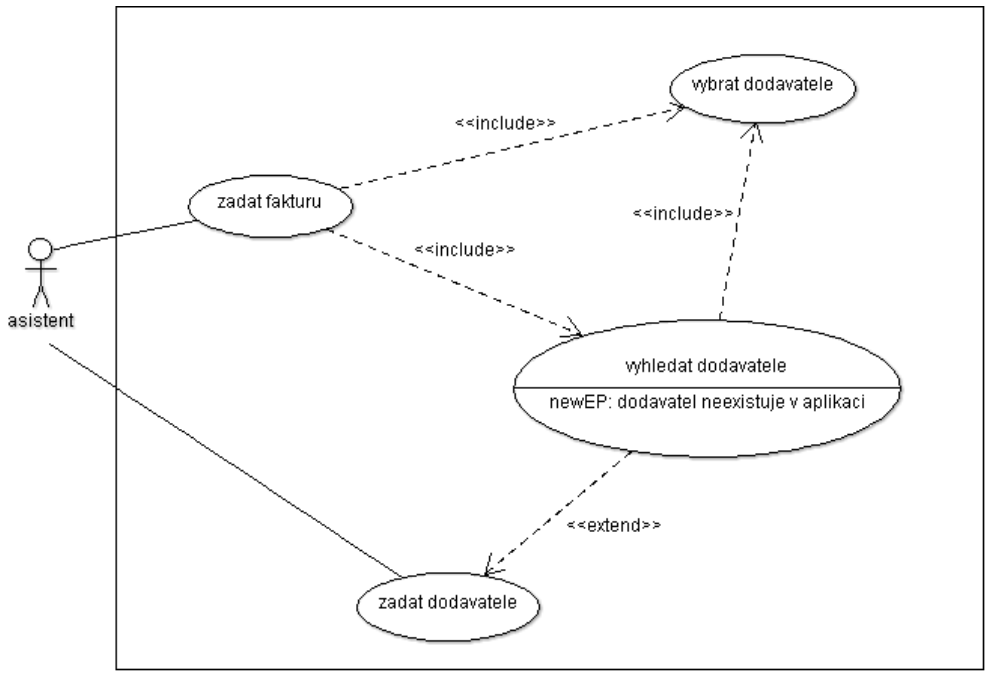
Jakmile budou tyto tři Use Case Diagramy hotové, seskupí se do jednoho Use Case Diagramu.

Jsou zde tři aktivní účastníci, a to asistent, účetní a hlavní účetní.

### 4.1.2 Use Case diagram – pro prvního účastníka: Asistenta

Z důvodu velikosti obrázku a místa umístění je tento diagram na následující stránce.

**Use Case diagram: Zadání faktury a dodavatele**




**Obrázek 23 - vlastní obrázek**

### 4.1.3 Popis Use Case diagramu:

Asistent(ka) se přihlašuje do aplikace, vyhledá dodavatele, vybere dodavatele, pokud není dodavatel, tak ho zadá. Zadá fakturu.

**Vstupní podmínky:**

1. asistent(ka) je přihlášen(a) do aplikace
2. faktury pro zadání do aplikace jsou již schváleny vedením firmy, či odpovědnou osobou; nemůže zadat do systému fakturu, která by nebyla schválena

**Tok událostí:**

1. začne zadávat fakturu
2. vyhledá dodavatele

3. pokud dodavatel existuje, vybere dodavatele
4. pokud dodavatel neexistuje, zadá dodavatele
5. dokončí zadání faktury

**Následné podmínky**

Po úspěšném zadání faktury je tato faktura připravena k zaúčtování.

**Alternativní tok**

V případě zadání stejné faktury dojde k blokaci jejího uložení.

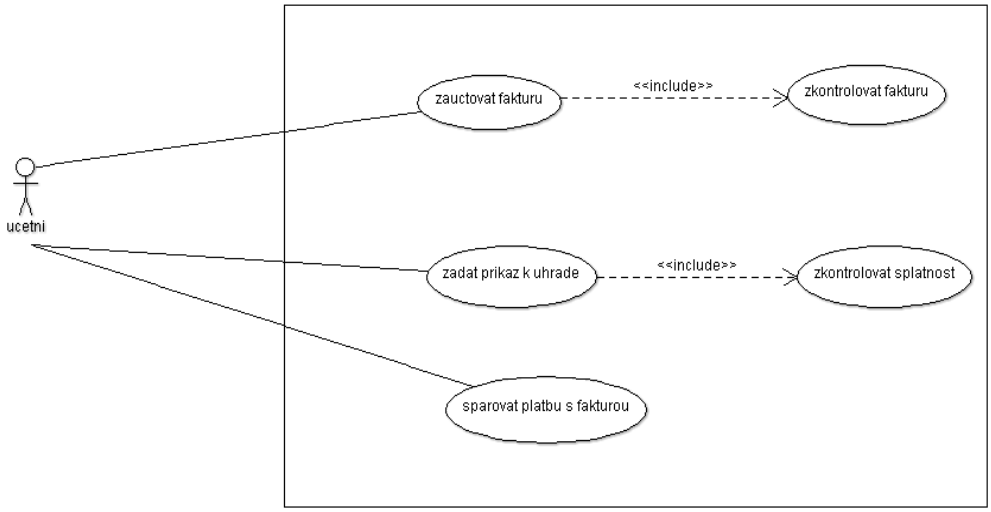
**Chybový tok**

Faktura je chybně zadána.

### 4.1.4 Use Case diagram pro druhého účastníka: Účetní

Dalším Use Case diagramem bude diagram pro účastníka účetní. Účetní má na starost kontrolu zadaných faktur, jejich zaúčtováním a provedení jejich plateb.

**Use Case Diagram: Kontrola, zaúčtování a platby faktur**



**Obrázek 24 - vlastní obrázek**

### 4.1.5 Popis Use Case diagramu pro účetní:

Účetní se přihlašuje do aplikace, zkontroluje fakturu, zaúčtuje fakturu.

Zkontroluje splatnost fakturu a následně vytvoří příkaz k úhradě. Po provedení platby a obdržení výpisu z bankovního účtu platbu spáruje s konkrétní fakturou.

Vstupní podmínky:

1. faktura musí být správně zadána

Tok událostí:

1. zkontroluje zadanou fakturu
2. zaúčtuje ji
3. zkontroluje splatnosti faktur
4. zpracuje příkaz k úhradě
5. spáruje platbu s konkrétní fakturou

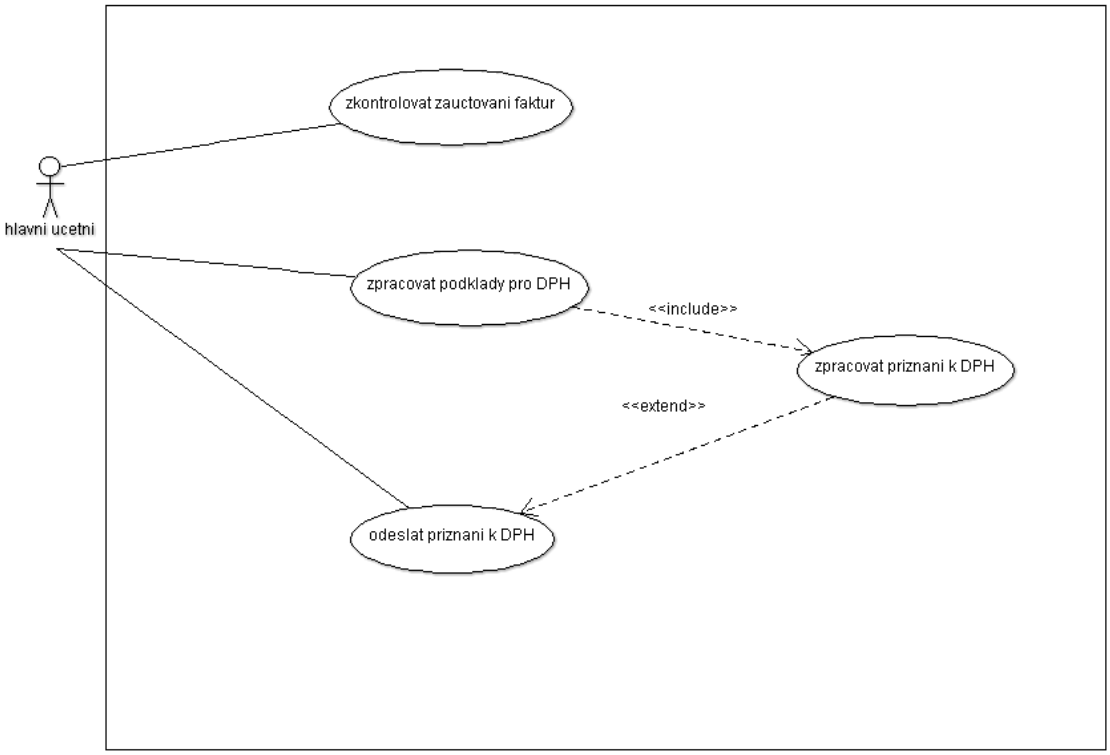
Následné podmínky
žádné

Alternativní tok
žádné

Chybový tok
Nastane v případě, že faktura je špatně zaúčtována.

### 4.1.6 Use Case diagram pro Hlavní účetní

Z důvodu velikosti a umístění obrázku je tento diagram umístěn na následující stránce.




**Obrázek 25 - vlastní obrázek**

### 4.1.7 Popis Use Case Diagramu pro Hlavní účetní:

Hlavní účetní se přihlašuje do aplikace, zkontroluje zauctovani faktur. Zpracuje DPH.

Vstupní podmínky:

1. faktura je správně zaúčtována

Tok událostí:

1. zkontroluje zaúčtování faktur
2. zpracuje podkladová data pro DPH
3. zpracuje přiznání k DPH
4. odešle přiznání k DPH příslušnému Finančnímu úřadu

Následné podmínky
Žádné

Alternativní tok
Žádné

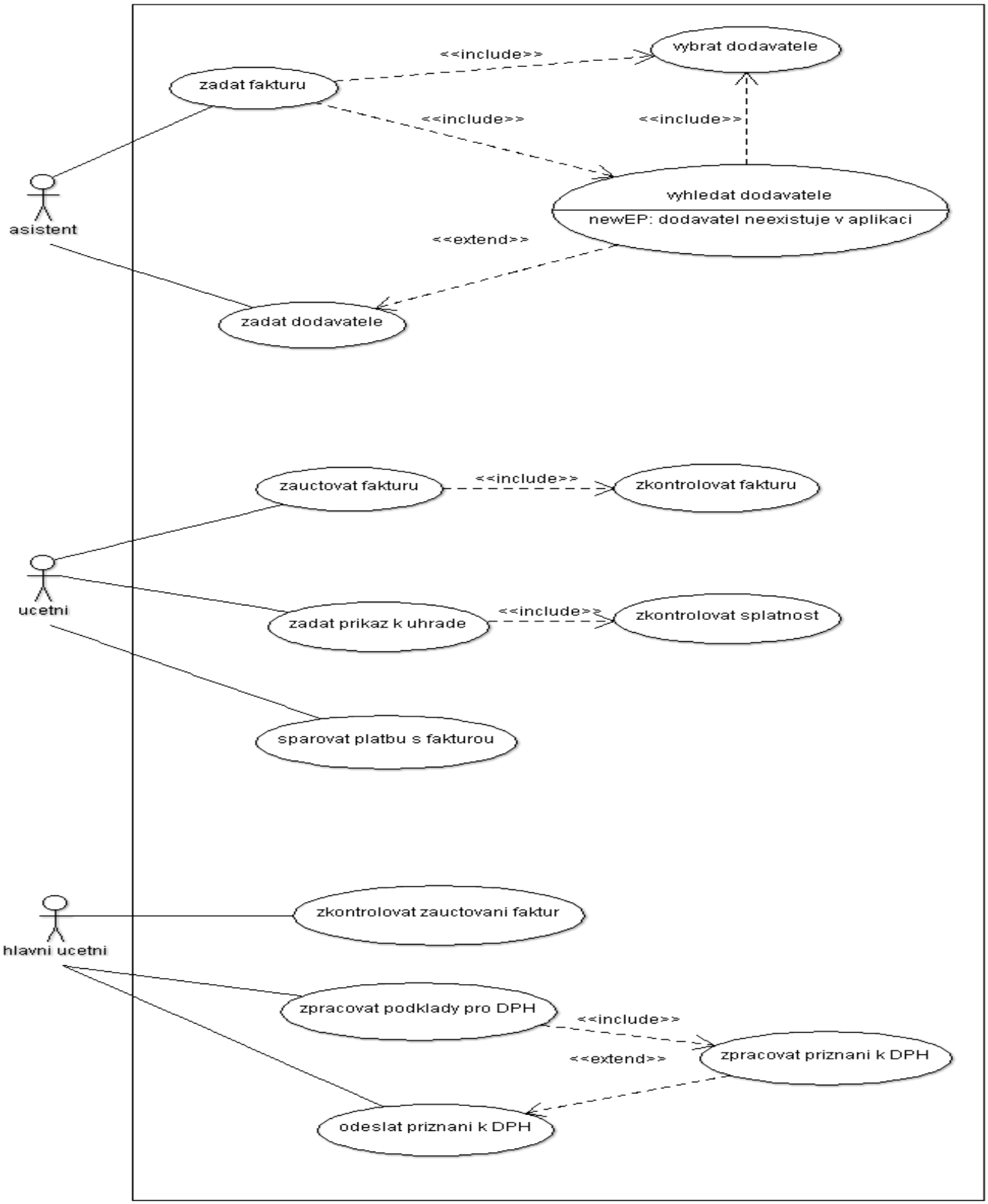
Chybový tok
Žádný

### 4.1.8 Celkový Use Case Diagram

Následně jsou seskupeny tyto tři Use Case diagramy do jednoho Use Case Diagramu.

Use Case Diagram pro zadání faktur, zadání dodavatele, zaúčtování faktur, kontroly zaúčtování a získání dat pro tvorbu DPH

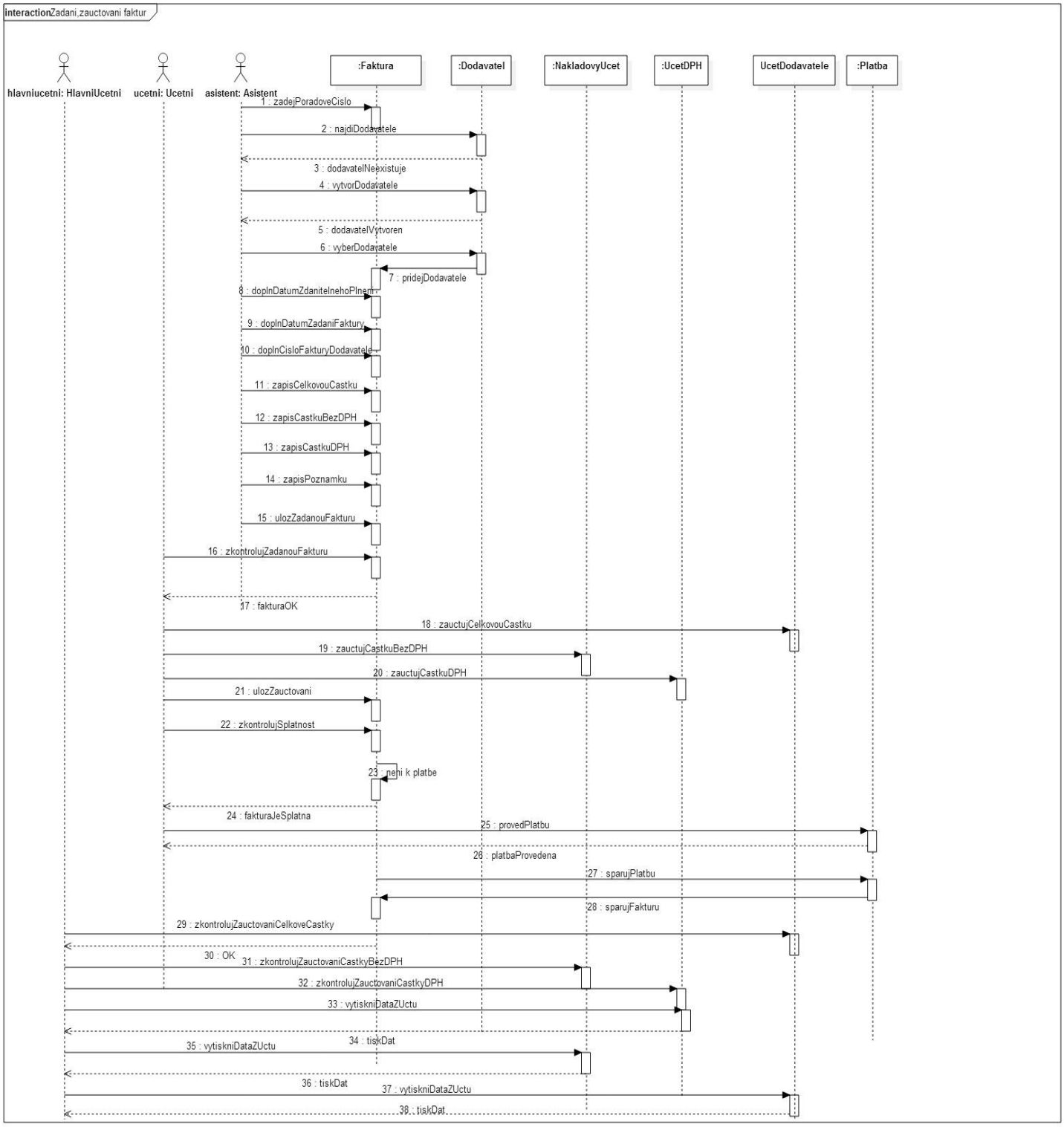
Z důvodu velikosti následujícího obrázku celkového Use Case Diagramu pro všechny účastníky bude tento diagram na následující stránce.




**Obrázek 26 - vlastní obrázek**

Po vytvoření Use Case Diagramů se může přikročit k tvorbě sekvenčního diagramu.

### 4.1.9 Sekvenční diagram




**Obrázek 27 - vlastní obrázek**

**Popis sekvenčního diagramu:**

Sekvenční diagram vychází z Use Case diagramu. Jsou zde též tři účastníci: asistent, účetní a hlavní účetní. Pak jsou zde objekty: faktura, dodavatel, platba, účet DPH a nákladový účet a účet dodavatele.

## 4.2 Návrh aplikace

Po analýze aplikace se přistoupí k samotnému návrhu aplikace. Do návrhu aplikace patří doménový model a diagram tříd.

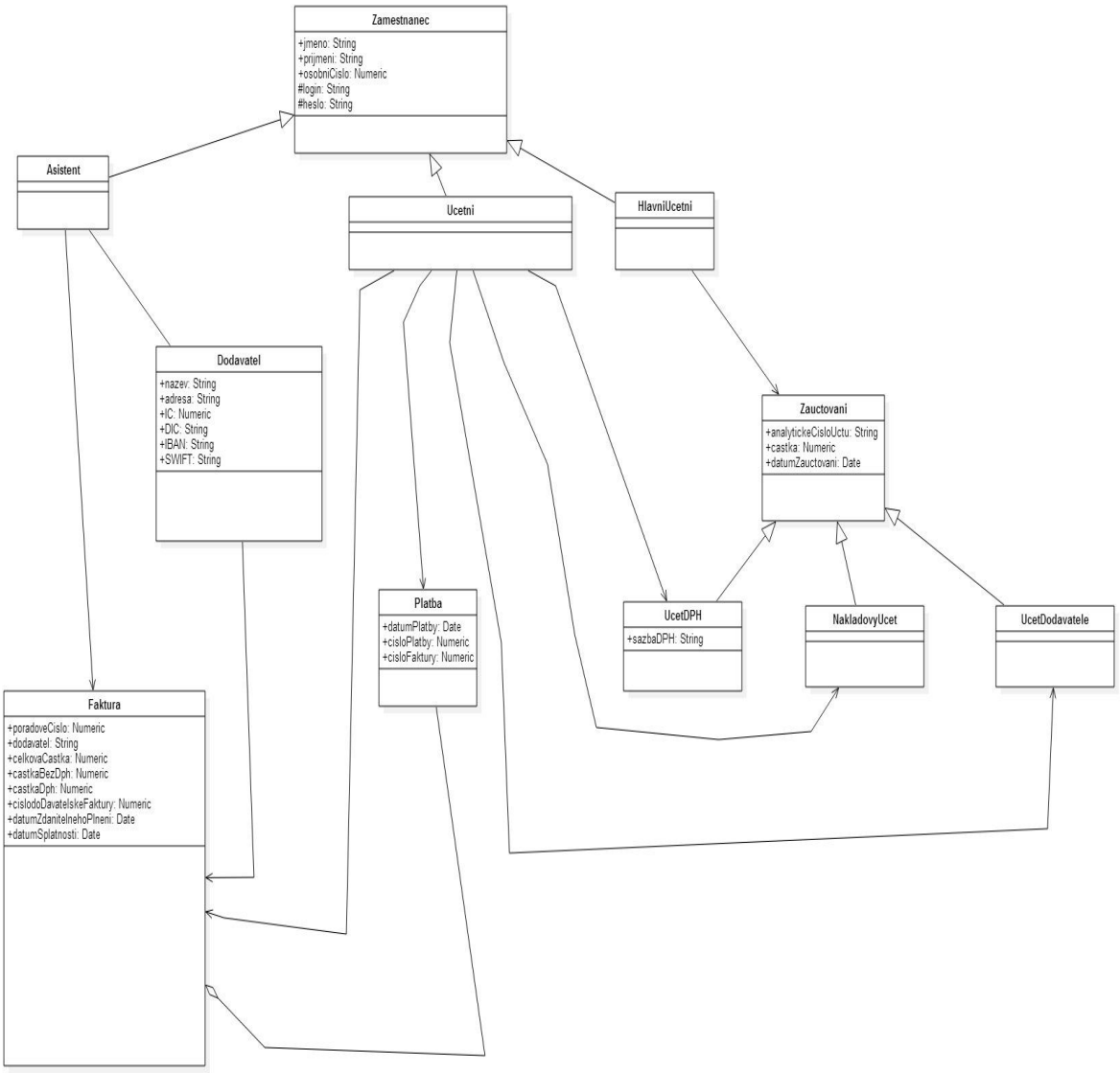
Doménový model je předlohou diagramu tříd. Neobsahuje metody a deklaraci datových typů. Pouze obsahuje třídy, jejich vazby mezi sebou a důležité atributy tříd.

### 4.2.1 Doménový model

Tento doménový model má 11 tříd. Nadtřídu Zaměstnance, ze které dědí podtřídy Asistent, Účetní a Hlavní účetní. Dále jsou zde třídy Dodavatel, Faktura, Platba. V tomto modelu je ještě jedna nadtřída, a to nadtřída Zaúčtování, ze které dědí další její podtřídy, a to třída ÚčetDPH, NákladovýÚčet a ÚčetDodavatele.

Z důvodu velikosti tohoto navrženého doménového modelu je tento model samostatně na další stránce této práce.

### 4.2.2 Doménový model



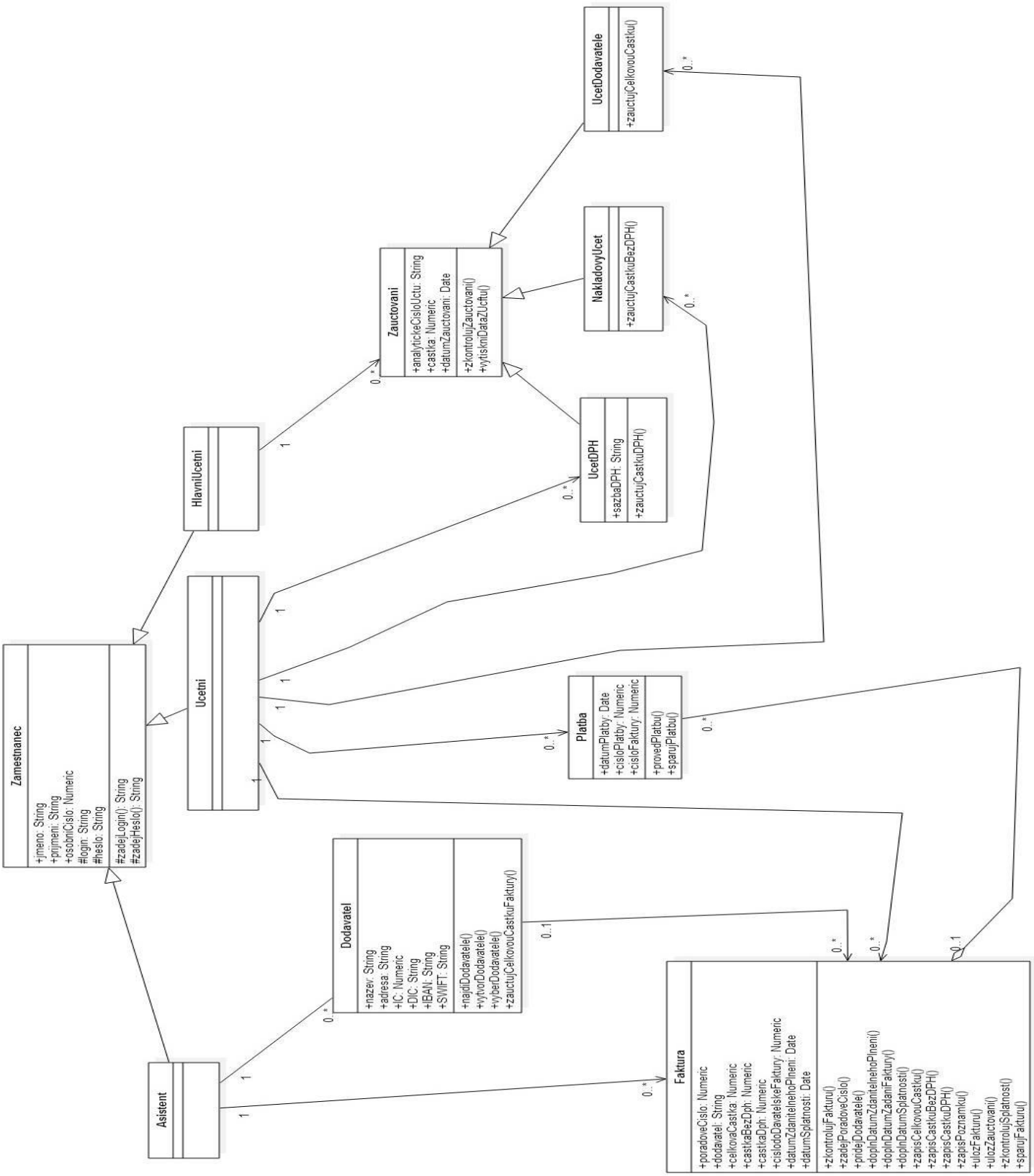

**Obrázek 28 - vlastní obrázek**

## 4.3 Implementace aplikace podle zásad softwarového inženýrství a standardu UML

Do implementace aplikace patří diagram tříd a pak vlastní kód aplikace.

Diagram tříd by měl být zvlášť pro každý programovací jazyk z důvodu různé syntaxe jazyka. Vzhledem k rozsahu této práce je tento diagram tříd pouze jeden všeobecný.

### 4.3.1 Diagram tříd



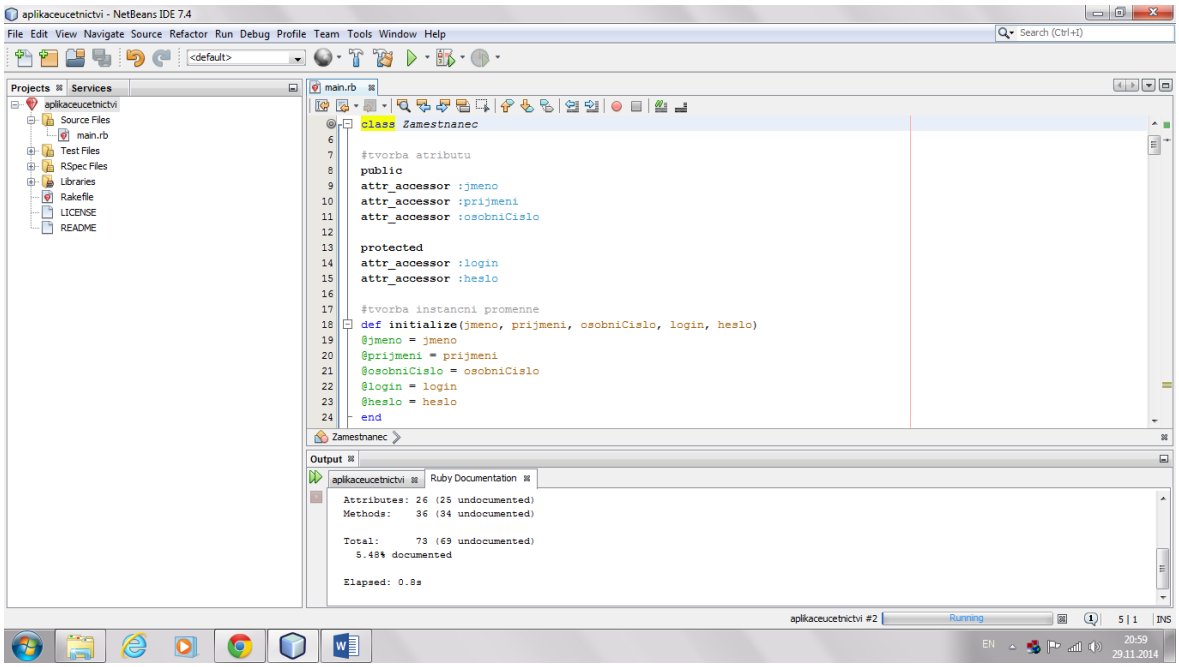

**Obrázek 29 - vlastní obrázek**

Tento konkrétní diagram tříd vychází z výše uvedeného doménového modelu.

Obsahuje nadtřídu zaměstnance, kde jejími podtřídami jsou třídy asistent(ka), účetní a hlavní účetní. Další třídou je třída dodavatel, faktura, platba, nadtřída zaúčtování, kde jejími podtřídami jsou třída nákladový účet a třída účet DPH a účet dodavatele.

Po vytvoření diagramu tříd se může přistoupit k samotnému programování.

### 4.3.2 Kód v programovacím jazyku Ruby v NetBeans IDE:




**Obrázek 30 - vlastní obrázek**

### 4.3.3 Vlastní zdrojový kód v programovacím jazyku Ruby

```
class Zamestnanec

#tvorba atributu
public
attr_accessor :jmeno
attr_accessor :prijmeni
attr_accessor :osobniCislo
```

```
protected
attr_accessor :login
attr_accessor :heslo

#tvorba instancni promenne
def initialize(jmeno, prijmeni, osobniCislo, login, heslo)
@jmeno = jmeno
@prijmeni = prijmeni
@osobniCislo = osobniCislo
@login = login
@heslo = heslo
end

#tvorba metody
def prihlas (login, heslo)
print "Zadej svuj login a heslo: #{@login} a #{@heslo} "
end

z = Zamestnanec.new("Hana", "Krizova", 3, @login, @heslo)
puts z.jmeno
puts z.prijmeni
puts z.osobniCislo

puts (z.prihlas = (@login=gets.strip, @heslo=gets.strip))

end


class Asistent < Zamestnanec
def initialize(jmeno, prijmeni, osobniCislo, login, heslo)
  super(jmeno, prijmeni, osobniCislo, login, heslo)
end
```

```
a = Asistent.new("Jan", "Novak", 25, @login, @heslo)

end

class Ucetni < Zamestnanec
def initialize(jmeno, prijmeni, osobniCislo, login, heslo)
  super(jmeno, prijmeni, osobniCislo, login, heslo)
end
end

class HlavniUcetni < Zamestnanec
def initialize(jmeno, prijmeni, osobniCislo, login, heslo)
  super(jmeno, prijmeni, osobniCislo, login, heslo)
end
end

class Dodavatel

public
attr_accessor :nazev
attr_accessor :adresa
attr_accessor :IC
attr_accessor :DIC
attr_accessor :SWIFT
attr_accessor :IBAN

@nazev = nazev
@adresa = adresa
@IC = IC
@DIC = DIC
@SWIFT = SWIFT
@IBAN = IBAN
```

```
def najdidodavatele
  find @nazev
  @nazev ==TRUE
  return


end

def vytvordodavatele
  create @nazev = nazev
  create @adresa = adresa
  create @IC = IC
  create @DIC = DIC
  create @SWIFT = SWIFT
  create @IBAN = IBAN
end

def vyberdodavatele
  select @nazev
  @nazev == TRUE
  return

end

def zauctujcelkovoucastkufaktury
end

end

class Faktura
attr_accessor :poradoveCislo
attr_accessor :dodavatel
```

```
attr_accessor :celkovaCastka
attr_accessor :castkaBezDph
attr_accessor :castkaDph
attr_accessor :cisloDodavatelskeFaktury
attr_accessor :datumZdanitelnehoPlneni
attr_accessor :datumSplatnosti

def initialize(poradoveCislo, dodavatel, celkovaCastka, castkaBezDph, castkaDph, cisloDodavatelskeFaktury,
  datumZdanitelnehoPlneni, datumsplatnosti)

  @poradoveCislo = poradoveCislo
  @dodavatel = dodavatel
  @celkovaCastka = celkovaCastka
  @castkaBezDph = castkaBezDph
  @castkaDph = castkaDph
  @cisloDodavatelskeFaktury = cisloDodavatelskeFaktury
  @datumZdanitelnehoPlneni = datumZdanitelnehoPlneni
  @datumSplatnosti = datumSplatnosti

end

def zkontrolujfakturu

end

def zadejporadovecislo
end

def pridejdodavatele
end
```

```
def doplndatumzdanitelnehoplneni
end

def doplndatumzadanifaktury
end

def doplndatumsplatnosti
end

def zapiscelkovoucastku
end

def zapiscastkubezdph
end

def zapiscastkudph
end

def zapispoznamku
end

def ulozfakturu
end

def ulozzauctovani
end

def zkontrolujsplatnost
end

def sparujfakturusplatbou
end
```

```
end

class Platba
attr_accessor :datumPlatby
attr_accessor :cisloPlatby
attr_accessor :cisloFaktury

def initialize(datumPlatby, cisloPlatby, cisloFaktury)
  @datumPlatby = datumPlatby
  @cisloPlatby = cisloPlatby
  @cisloFaktury = cisloFaktury
end

def provedplatbu
end

def sparujplatbusfakturou
end

end

class Zauctovani

attr_accessor :analytickeCisloUctu
attr_accessor :castka
attr_accessor :datumZauctovani

def initialize(analytickeCisloUctu, castka, datumZauctovani)
  @analytickeCisloUctu = analytickeCisloUctu
  @castka = castka
  @datumZauctovani = datumZauctovani
end
```

```
def zkontrolujzauctovani
end

def vytisknidatazuctu
end
end


class UcetDPH < Zauctovani

attr_accessor :sazbaDph

def initialize(analytickeCisloUctu, castka, datumZauctovani, sazbaDph)
  @sazbaDph = sazbaDph
  super(analytickeCisloUctu, castka, datumZauctovani)
end


def zauctujcastkudph
end

end

class NakladovyUcet < Zauctovani
def initialize(analytickeCisloUctu, castka, datumZauctovani)
  super(analytickeCisloUctu, castka, datumZauctovani)
end

def zauctujcastkubezdph
end
  end
class UcetDodavatele < Zauctovani
```

```
def initialize(analytickeCisloUctu, castka, datumZauctovani)
  super(analytickeCisloUctu, castka, datumZauctovani)
end

def zauctujcelkovoucastku
end

end
```

### 4.3.4 Vygenerovaná dokumentace k vytvořenému kódu v programovacím jazyku Ruby

NetBeans pro programovací jazyk generuje dokumentaci.

RDoc Documentation

This is the API documentation for 'RDoc Documentation'.

Files

LICENSE

README

Rakefile

rake-d.txt

Classes/Modules

Asistent

Dodavatel

Faktura

HlavniUcetni

NakladovyUcet

Platba

UcetDPH

UcetDodavatele

Ucetni

Zamestnanec

Zauctovani

Methods

::new — Zamestnanec

::new — Platba

::new — Asistent

::new — Ucetni

::new — HlavniUcetni

::new — Zauctovani

::new — Faktura

::new — NakladovyUcet

::new — UcetDodavatele

::new — UcetDPH

#doplndatumsplatnosti — Faktura

#doplndatumzadanifaktury — Faktura

#doplndatumzdanitelnehoplneni — Faktura

#najdidodavatele — Dodavatel

#pridejdodavatele — Faktura

#prihlas — Zamestnanec

#provedplatbu — Platba

#sparujfakturusplatbou — Faktura

#sparujplatbusfakturou — Platba

#ulozfakturu — Faktura

#ulozzauctovani — Faktura

#vyberdodavatele — Dodavatel

#vytisknidatazuctu — Zauctovani

#vytvordodavatele — Dodavatel

#zadejporadovecislo — Faktura

#zapiscastkubezdph — Faktura

#zapiscastkudph — Faktura

#zapiscelkovoucastku — Faktura

#zapispoznamku — Faktura

#zauctujcastkubezdph — NakladovyUcet

#zauctujcastkudph — UcetDPH

#zauctujcelkovoucastku — UcetDodavatele

#zauctujcelkovoucastkufaktury — Dodavatel

#zkontrolujfakturu — Faktura

#zkontrolujsplatnost — Faktura

#zkontrolujzauctovani — Zauctovani

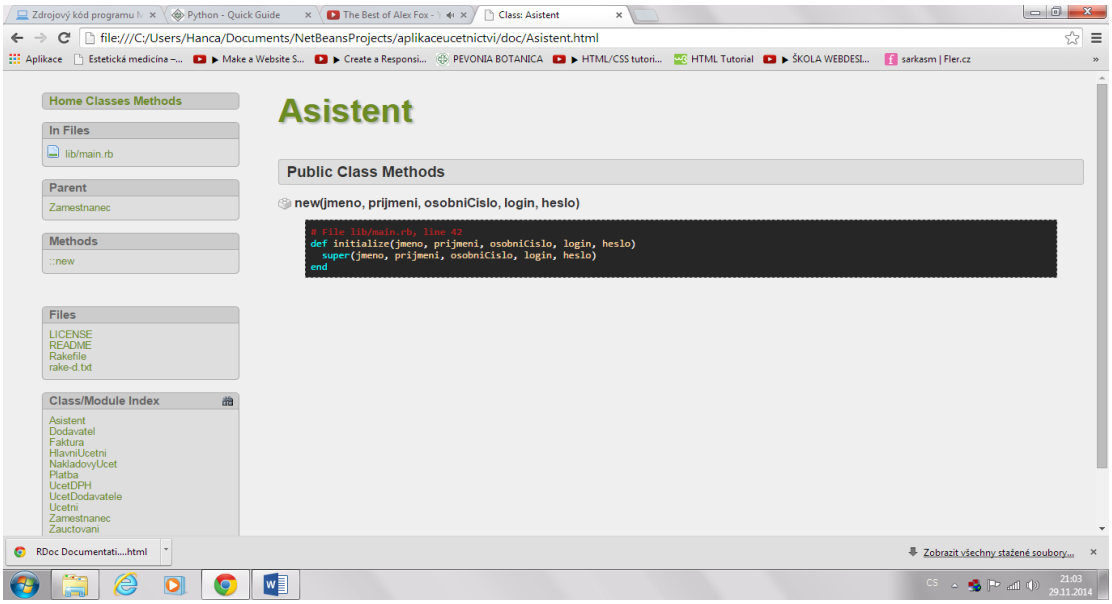
[Validate]

Generated with the Darkfish Rdoc Generator 2.

### 4.3.5 Ilustrace třídy Asistent v NetBeans IDE (Ruby)

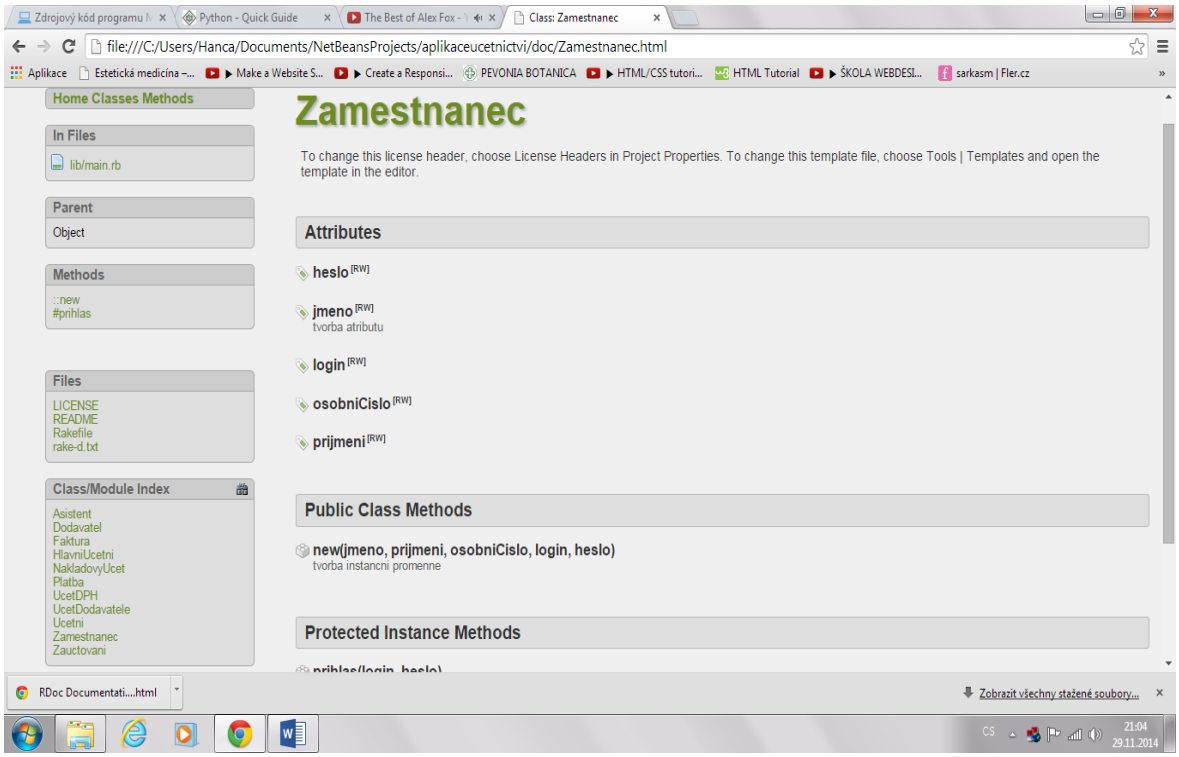
Pro ilustraci takto vypadá třída Asistent. Její nadtřídou je třída Zaměstnanec. Také je možné vidět, jaké atributy a metody obsahuje tato třída.

Z důvodu velikosti obrázku, je tento obrázek na další stránce.



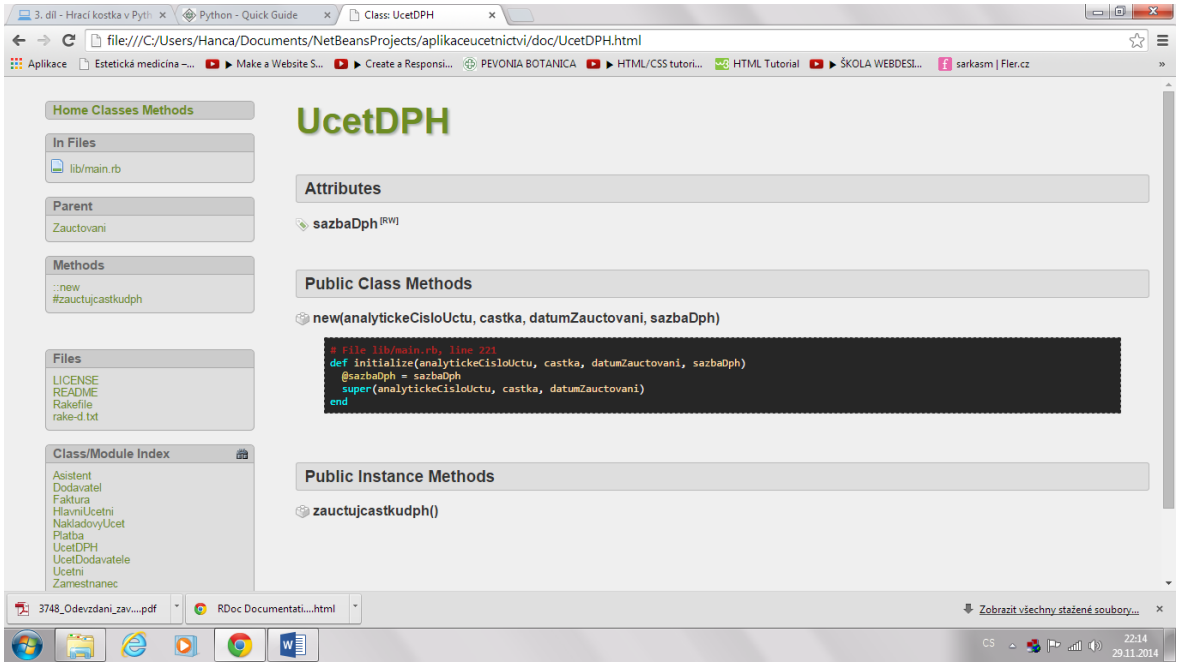

**Obrázek 31 - vlastní obrázek**

### 4.3.6 Ilustrace třídy Zaměstnanec v NetBeans IDE (Ruby)




**Obrázek 32 - vlastní obrázek**

### 4.3.7 Ilustrace třídy Účet DPH v NetBeans IDE (Ruby)




**Obrázek 33 - vlastní obrázek**

### 4.3.8 Kód v programovacím jazyku Python v NetBeans IDE:

NetBeans nemá pro tento programovací možnost generování dokumentace

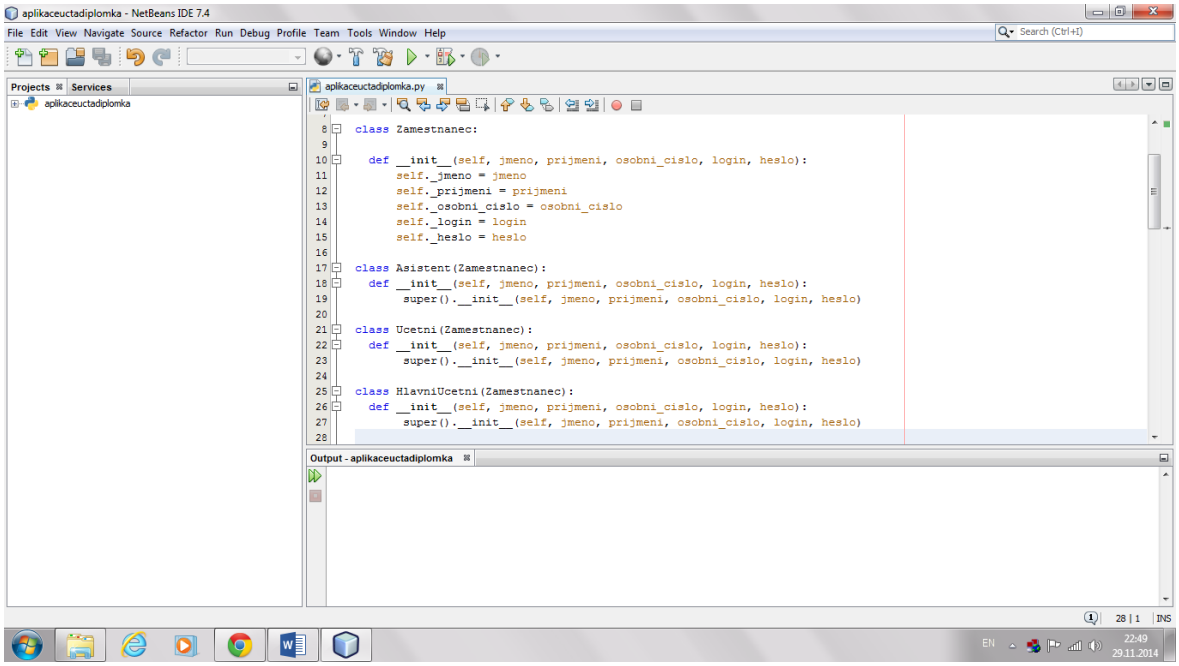
```
class Zamestnanec:

  def __init__(self, jmeno, prijmeni, osobni_cislo, login, heslo):
      self._jmeno = jmeno
      self._prijmeni = prijmeni
      self._osobni_cislo = osobni_cislo
      self._login = login
      self._heslo = heslo

class Asistent(Zamestnanec):
  def __init__(self, jmeno, prijmeni, osobni_cislo, login, heslo):
```

```
        super().__init__(self, jmeno, prijmeni, osobni_cislo, login, heslo)

class Ucetni(Zamestnanec):
  def __init__(self, jmeno, prijmeni, osobni_cislo, login, heslo):
        super().__init__(self, jmeno, prijmeni, osobni_cislo, login, heslo)

class HlavniUcetni(Zamestnanec):
  def __init__(self, jmeno, prijmeni, osobni_cislo, login, heslo):
        super().__init__(self, jmeno, prijmeni, osobni_cislo, login, heslo)

class Dodavatel:
  def __init__(self, nazev, adresa, IC, DIC, IBAN, SWIFT):
      self.__nazev = nazev
      self.__adresa = adresa
      self.__IC = IC
      self.__DIC = DIC
      self.__IBAN = IBAN
      self.__SWIFT = SWIFT

class Faktura:
  def __init__(self, poradove_cislo, dodavatel, celkova_castka, castka_bez_dph,
  castka_dph, cislo_dodavatelske_faktury, datum_zdanitelneho_plneni, datum_splatnosti):
      self.__poradove_cislo = poradove_cislo
      self.__dodavatel = dodavatel
      self.__celkova_castka = celkova_castka
      self.__castka_bez_dph = castka_bez_dph
      self.__castka_dph = castka_dph
      self.__cislo_dodavatelske_faktury = cislo_dodavatelske_faktury
      self.__datum_zdanitelneho_plneni = datum_zdanitelneho_plneni
      self.__datum_splatnosti = datum_splatnosti

class Platba:
```

```
    def __init__(self, datum_platby, cislo_platby, cislo_faktury):
        self.__datum_platby = datum_platby
        self.__cislo_platby = cislo_platby
        self.__cislo_faktury = cislo_faktury


class Zauctovani:
    def __init__(self, analyticke_cislo_uctu, castka, datum_zauctovani):
        self.__analyticke_cislo_uctu = analyticke_cislo_uctu
        self.__castka = castka
        self.__datum_zauctovani = datum_zauctovani

class UcetDPH(Zauctovani):
    def __init__(self, analyticke_cislo_uctu, castka, datum_zauctovani, sazba_DPH):
      super().__init__(self, analyticke_cislo_uctu, castka, datum_zauctovani)
      self.__sazba_DPH = sazba_DPH

class NakladovyUcet(Zauctovani):
    def __init__(self, analyticke_cislo_uctu, castka, datum_zauctovani):
       super().__init__(self, analyticke_cislo_uctu, castka, datum_zauctovani)


class UcetDodavatele(Zauctovani):
    def __init__(self, analyticke_cislo_uctu, castka, datum_zauctovani):
       super().__init__(self, analyticke_cislo_uctu, castka, datum_zauctovani)
```

Níže je uvedený obrázek z NetBeans během programování v programovacím jazyku Python.

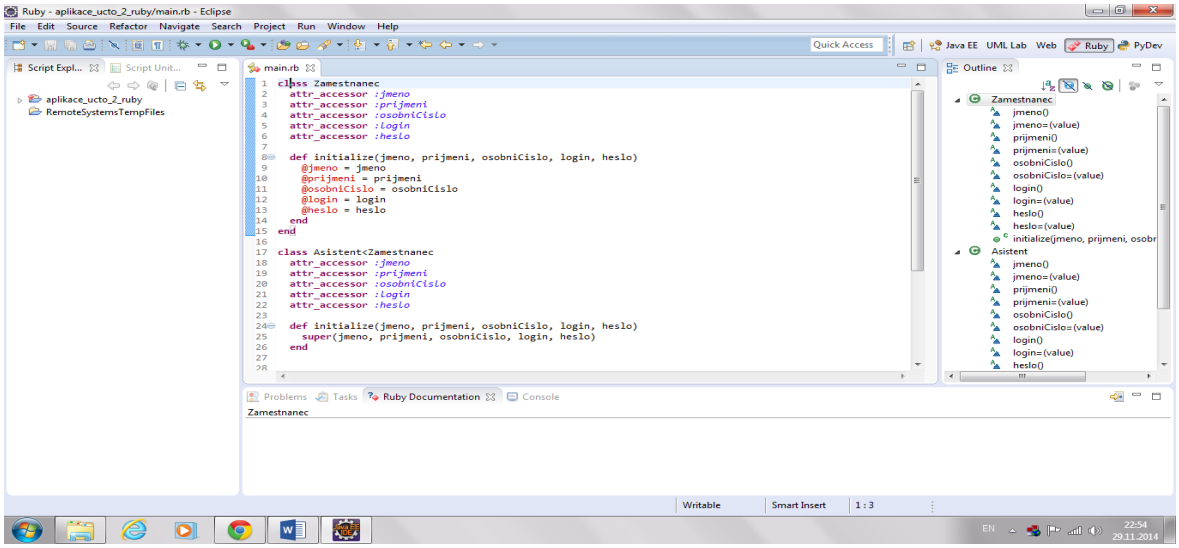
V levé části obrazovky můžeme vidět otevřený projekt pro programování v Pythonu. V pravé části vidíme kód a v dolní části by byly vidět výsledky běžícího programu.




**Obrázek 34 - vlastní obrázek**

### 4.3.9 Kód v programovacím jazyku Ruby v Eclipse IDE:

Zde nebyla nijak změněna syntaxe jazyka. Proto je níže uvedený obrázek pro ilustraci prostředí Eclipse. V levé části obrazovky je možné vidět otevřené projekty, ve střední části obrazovky je kód aplikace a v pravé části obrazovky jsou vidět výstupy, tzn. názvy tříd, jejich atributy, metody atd.




**Obrázek 35 - vlastní obrázek**

## 4.4 Porovnání programovacích jazyků Python a Ruby

| Vlastnost (popis) | Ruby | Rozdíl | Python |
|---|---|---|---|
| definice třídy | class Jmeno<br>tělo třídy<br>end | Python má za názvem třídy dvojtečku a třída není ukončena slovem „end“ | class Jmeno: |
| Dědění; zápis podtřídy | class Ucetni<Zamestnanec | Nadtřída u Pythonu je v kulatých závorkách | class Ucetni(Zamestnanec): |
| hlavni soubor<br>main.rb<br>main.py | při tvorbě zadání projektu nebo už již ve vytvořeném projektu, pokud nebyl vytvořen v úvodu | žádný rozdíl, pouze mezi prostředími NetBeans a Eclipse. Hlavní soubor v Eclipse je tvořen až ve vytvořeném projektu. | při tvorbě zadání projektu nebo už již ve vytvořeném projektu, pokud nebyl vytvořen v úvodu |
| Komentář | #komentář | žádný | #komentář |
| proměnné a datové typy | proměnné nemají typy, ale objekty mají | | datové typy si proměnné deklarují samy, protože Python je dynamickým jazykem |
| metoda puts | vypíše hodnoty proměnných na více řádků | žádný | vypíše hodnoty proměnných na více řádků |
| metoda print | vypíše hodnoty proměnných na jeden řádek | žádný | vypíše hodnoty proměnných na jeden řádek |
| poziční argument (self) | Ne | existence | ano |
| proměnná třídy | @@proměnná | | proměnná přidružená k třídě, ne k instanci třídy |
| proměnná instance | @proměnná | | instance.promenna = hodnota |

# 5 Závěr

Cílem práce bylo vybrat několik objektových jazyků a prostředí, reprezentujících různé přístupy v objektové tvorbě softwaru, porovnat jejich vlastnosti a demonstrovat je na příkladu funkční aplikace části účetnictví.

Tohoto cíle bylo dosaženo:

Za prvé, v teoretické části rozborem jednotlivých objektových jazyků a prostředích NetBeans IDE a Eclipse.

Za druhé, tento cíl byl dosažen konkrétní analýzou, návrhem a implementací konkrétní aplikace pro účetnictví. V analýze byly rozebrány jednotlivé požadavky zákazníka a navrženy Use Case diagramy pro jednotlivé účastníky této aplikace a pak byly tyto Use Case diagramy spojeny do jednoho Use Case diagramu. Po Use Case diagramu následovalo vytvoření sekvenčního diagramu. V návrhu byl vytvořen doménový diagram, který byl předlohou pro tvorbu diagramu tříd v části implementace aplikace.

V části implementace byly vytvořeny kódy pro tuto aplikaci v programovacích jazycích Ruby a Python ve vývojovém prostředí NetBeans IDE 7.4 a byl naznačen kód i ve vývojovém prostředí Eclipse.

V závěrečné části byly porovnány vlastnosti těchto dvou objektových programovacích jazyků.

Zadaný cíl v úvodu tato práce splnila.

# 6 Seznam použitých zdrojů

## 6.1 Literatura

MERUNKA, Vojtěch. *Objektové modelování.* 1. vydání Praha 2: Alfa Nakladatelství, s.r.o., 2008 s.197 ISBN 978-80-87197-04-2

ČADA, Ondřej. *Objektové programování naučte se pravidla objektového myšlení.* 1. vydání Praha: Grada Publishing, a.s. 2009 s.200 ISBN 978-80-247-2745-5

KEOGH, Jim, GIANNINI, Mario. *OOP bez předchozích znalostí. Průvodce pro samouky.* Překlad: Veronika Matějů, dotisk 1. vydání Brno: Computer Press, a.s. 2010 s. 222 ISBN 80-251-0973-9

FULTON, Hal. *RUBY kompendium znalostí pro začátečníky i profesionály.* 1. vydání Brno: ZONER software, s.r.o. 2009 s.769 ISBN 978-80-7413-018-2

SUMMERFIELD, Mark. *Python 3 Výukový kurz.* 1. vydání Brno: Computer Press, a.s. 2010 s.584 ISBN 978-80-251-2737-7

## 6.2 Elektronické zdroje:

WIKIPEDIA
http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD

WIKIPEDIA
http://cs.wikipedia.org/wiki/Polymorfismus_(programov%C3%A1n%C3%AD)

GUIDES RUBYONRAILS
http://guides.rubyonrails.cz/

APPLE

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

MOZILLA

https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

NETBEANS

https://netbeans.org/features/index.html

ECLIPSE

https://eclipse.org/org/

# 7 Příloha A

V příloze A jsou uvedeny originály textů v anglickém jazyce, které jsou volně přeloženy v teoretické části této diplomové práce. Zdoje jsou uvedeny v žádsti přehledu zdrojů v části elektronických podkladů.

***Objective C – originál znění textu – zdroj: již uvedený odkaz***

*Objective-C is the primary programming language you use when writing software for OS X and iOS. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods.*

*Objective-C uses protocols to define a group of related methods.*

*The Objective-C syntax used to declare a class interface looks like this:*

```
@interface SimpleClass : NSObject


@end
```

*This example declares a class named SimpleClass, which inherits from NSObject.*

*Xcode, Apple's integrated development environment (IDE) for creating OS X and iOS software.*

*In Objective-C, a class is itself an object with an opaque type called Class. Classes can't have properties defined using the declaration syntax shown earlier for instances, but they can receive messages.*

***Javascript– originál znění textu – zdroj: již uvedený odkaz***

*JavaScript can function as both a procedural and an object oriented language. Objects are created programmatically in JavaScript, by attaching methods and properties to otherwise empty objects at run time, as opposed to the syntactic class definitions common in compiled languages like C++ and Java. Once an object has been constructed it can be used as a blueprint (or prototype) for creating similar objects.*

*JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name and a value. A property's value can be a function, in which case the property is known as a method. In addition to objects that are predefined in the browser, you can define your own objects.*

***NetBeans IDE– originál znění textu – zdroj: již uvedený odkaz***
*NetBeans IDE lets you quickly and easily develop Java desktop, mobile, and web applications, as well as HTML5 applications with HTML, JavaScript, and CSS. The IDE also provides a great set of tools for PHP and C/C++ developers. It is free and open source.*

*An IDE is much more than a text editor. The NetBeans Editor indents lines, matches words and brackets, and highlights source code syntactically and semantically. It also provides code templates, coding tips, and refactoring tools.*

*The editor supports many languages from Java, C/C++, XML and HTML, to PHP, Groovy, Javadoc, JavaScript and JSP. Because the editor is extensible, you can plug in support for many other languages.*

*NetBeans IDE can be installed on all operating systems that support Java, from Windows to Linux to Mac OS X systems.*

*NetBeans IDE is a modular developer tool for a wide range of application development technologies. The base IDE includes an advanced multi-language editor, Debugger and Profiler, as well as tools for versioning control and developer collaboration.*

*NetBeans IDE gives you skeleton applications in the form of project templates for all the technologies it supports. In addition, it provides a set of sample applications, some of which can be recreated step by step by following a related tutorial available on NetBeans.org.*

*The IDE provides project templates and sample projects that help you create Java SE applications, Java EE applications, Java ME applications, HTML5 applications, NetBeans Platform applications, PHP application, and C/C++ applications.*

***Eclipse IDE– originál znění textu – zdroj: již uvedený odkaz***
*Eclipse is famous for our Java Integrated Development Environment (IDE), but our C/C++ IDE and PHP IDE are pretty cool too. You can easily combine language support and other features into any of our default packages, and the Eclipse Marketplace allows for virtually unlimited customization and extension.*