Pedagogická fakulta
Faculty of Education
Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky

Bakalářská práce

WYSIWYG editor pro výuku OOP v Javě

Vypracoval: Adam Chovanec
Vedoucí práce: RNDr. Hana Havelková

České Budějovice 2015

PROHLÁŠENÍ

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích

dne: ....

Podpis studenta

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Adam CHOVANEC**
Osobní číslo:	**P11044**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Název tématu:	**WYSIWYG editor pro výuku OOP v Javě**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Cílem práce je vytvořit WYSIWYG Java editor - speciální prostředí pro podporu výuky objektově orientovaného programování v Javě, který umožní uživateli snazší zvládnutí základních principů OOP konkrétně v programovacím jazyce Java.
Editor by měl umožnit oboustrannou editaci - jak psaní přímého kódu, tak i grafickou tvorbu kódu - programování pomocí speciálních grafických prvků (komponent), díky čemuž by mělo být programování jednodušší a speciálně pro začátečníky méně náročné.
Aplikace bude nabízet uživateli modelovat strukturu tříd, metod, konstruktorů; nastavit modifikátory přístupu; určovat návratové typy. K dispozici budou též šablony základních jazykových struktur.

Rozsah grafických prací: **CD ROM**

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **HEROUT, P. Učebnice jazyka Java. Praha : Kopp, 2002. ISBN 80-7232-115-3.**
2. **HEROUT, P. Java - grafické uživatelské prostředí a čeština. České Budějovice: Kopp, 2007.**
3. **ECKEL, B. Thinking in Java, 3rd edition. ¡http://www.mindview.net/Books/TIJ/.**
4. **BARNES, D., J., KÖLLING, M. Objects First with Java A Practical Introduction using BlueJ, 5th edition. Prentice Hall / Pearson Education, 2012. ISBN 978-013-249266-9.**
5. **ORACLE. Java Platform Standard Edition 7 Documentation, http://docs.oracle.com/javase/7/docs/index.html.**

Vedoucí bakalářské práce: **RNDr. Hana Havelková**
Katedra informatiky

Datum zadání bakalářské práce: **16. dubna 2013**

Termín odevzdání bakalářské práce: **30. dubna 2014**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 16. dubna 2013

ANOTACE

Tato práce je o vytvoření výukového programu, který umožní uživateli lépe porozumět principům OOP a Javě samotné. Jedná se spíše o praktickou část práce, kde výstupem bude samotný program.

Hlavní složkou tohoto programu je intuitivnost a možnost tvořit si za pomoci grafických prvků vlastní struktury. Teoretická část by se zaobírala principy OOP.
V současné době není moc programů pro začínající programátory, ale co je hlavní, nedokážou myslet objektově. Proto by mohl právě tento program pomoci přimět studenty myslet jinak.

Klíčová slova:

WYSIWYG editor, UML, program, výuka, Java, objektově orientované programování, ICT, grafické uživatelské prostředí

ABSTRACT

This bachelor work is about to create an educational program, which allow user to better understand the basic principles of object oriented programming and Java itself.It is rather the practical part of this work, where output will be program itself.

Main aspect of this program is intuitiveness and possibility to create own object structures for using graphical elements. In the theoretical part will be mentioning the basic principles of object oriented programming.

Nowadays it's not much programs for novice programmer, but what is the main problem, that they don't understand OOP. Therefore this program can help students to think differently.

Keywords:

WYSIWIG editor, UML, program, teaching, Java, Object Oriented Programming, ICT, graphic user interface

PODĚKOVÁNÍ

Rád bych touto cestou poděkoval své vedoucí práce RNDr. Haně Havelkové za cenné rady a doporučení při vedení mé bakalářské práce.

Děkuji rovněž svým kamarádům a mé testovací skupině, ti všichni mi pomohli při ověřování aplikace a poskytli mi tak vzácnou zpětnou vazbu.

Na závěr bych chtěl za podporu při studiu poděkovat především své rodině a blízkým přátelům.

OBSAH

1 Úvod

V dnešní době se stále více a více lidí snaží proniknout do světa programování. Stěžejním rozhodnutím pro začínajícího programátora je však výběr vhodného programovacího jazyka. Střední a vysoké školy pro tento účel volí programovací jazyk Java, který nabízí srozumitelnou syntaxi a jednoduchý přístup k objektovým principům.

Díky tomu, jak se Java stává velmi rozšířeným programovacím jazykem, začíná se čím dál tím více vyučovat na školách. Tato volba s sebou však přináší jistá rizika. Jelikož Java je objektově orientovaný jazyk, má spousta studentů potíže porozumět základním principům objektově orientovaného programování (dále jen OOP). Dalším problémem se stává, že mnoho programátorů spíše programuje, než provádí celkový návrh svého projektu.

Pro lepší pochopení a znázornění chování daného jazyka, či jeho principů se využívají speciální programy, které dokážou tuto problematiku osvětlit. Nicméně těchto programů není mnoho a často jsou to spíše vývojová prostředí, která pouze usnadní programování.

Prostřednictvím této bakalářské práce bych rád vytvořil aplikaci, která by byla schopna jednoduše a rychle navrhovat Java projekty a zároveň osvětlila a dopomohla k porozumění základním principům Javy.

1.1 Problematika

Jelikož je programování v Javě dnes velmi populární a řada škol se snaží Javu prosazovat i ve své výuce, vznikla velká poptávka po znalosti tohoto jazyka. Spousta studentů se snaží proniknout do světa OOP, avšak nedaří se jim pochopit základní principy.

Sám jsem tento jazyk studoval a zajímal se o něj, a proto vím o určitých úskalích, která tento jazyk přináší. Taktéž jsem působil jako lektor pro své vrstevníky, kteří s Javou bojovali a nedokázali pochopit některé její části, či terminologii. Od nich jsem se dozvěděl, které z těchto částí dělají studentům největší problémy.

Spousta mých „žáků“ se mne ptala, jestli existuje nějaký software, který by jim pomohl lépe porozumět problematice. To mě přimělo zamyslet se nad tím, že těchto aplikací není mnoho. Spousta těchto programů jsou buď profesionální vývojová prostředí, nebo výukové programy (viz *kapitola 3* Analýza).

Rozhodl jsem se proto vytvořit aplikaci, která by měla studentům usnadnit pochopení principů Javy a dopomoci jim při studiu tohoto jazyka.

1.2 Cíle práce

Cílem této bakalářská práce je zhotovit aplikaci, která bude podporou pro začínající programátory v jazyce Java. Při vývoji aplikace by měl být kladen důraz na přívětivost a jednoduchost v oblasti ovládání.

Dalším milníkem by měla být jistá forma vizualizace OOP aspektů jazyka Java, a to formou grafických prvků. Uživatel si tak bude moci lépe představit, za čím se skrývá objektově orientované programování.

Aplikace se bude skládat ze dvou hlavních částí. V modelovací části bude uživatel schopen navrhovat strukturu projektu a vytvářet tak svůj vlastní koncept. K tomuto účelu mu budou sloužit grafické komponenty, které mohou být dále upravovány a rozšiřovány.

Část programovací zajistí editor kódu sloužící k vlastnímu programování navržených komponent. Uživatel si bude moci doprogramovat navržené třídy v jazyce Java, kde samotná aplikace bude dohlížet na korektnost kódu.

Uživatelem navržený projekt bude možné uložit ve formě souboru a posléze ho i opětovně načíst. Dalším z možností výstupu aplikace bude export vymodelovaných tříd do zdrojových souborů jazyka Java.

Součástí práce bude poskytnutí řady názorných příkladů vytvořených v aplikaci a technická dokumentace.

1.3 Metoda práce

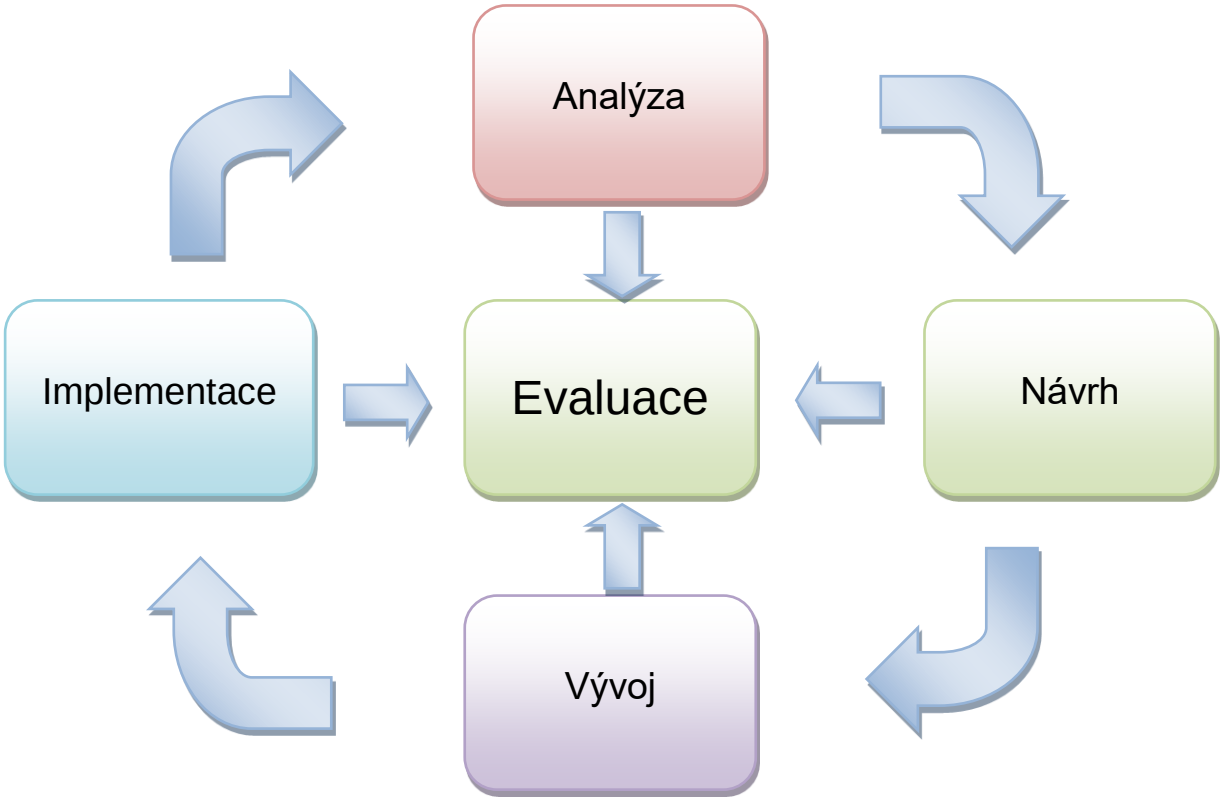
Pro vytváření komplexnějších aplikací se využívají různé metody, jak takový software vytvořit. Jeden z nejpopulárnějších a nejpoužívanějších přístupů při vytváření software se stává tzv. ADDIE model [1], který v pěti základních fázích definuje efektivní metodiku postupu. Při tvorbě této práce byl použit právě tento model, který tvoří strukturu z jednotlivých fází (Obr. 1-1).

V první řadě byla provedena analýza práce, která zahrnovala, co dané téma obnáší a jaké prostředky a materiály se budou využívat. Byla zvážena možná rizika a styl programování daného programu.

Dalším milníkem analýzy bylo zjistit vhodnost dostupných výukových software, které se zaměřují na výuku Javy, či dopomáhají k usnadnění při tvorbě jednoduchých aplikací. Na

základě zjištěných výsledků a vlastních poznatků od studentů byl proveden návrh samotné aplikace.

Ve fázi návrhu byl proveden hrubý náčrt toho, jak daná aplikace bude vypadat a jaké funkce bude nabízet. Dále byl sestrojen základní wireframe[1] prvotního vzhledu aplikace a byl sestaven postup při vývoji jádra[2] aplikace.



Obr. 1-1 Znázornění ADDIE Modelu

Vývojová fáze tedy zahrnovala samotné vytvoření aplikace a rozdělení vývoje do menších milníků. Tyto milníky poté tvořily jednotlivé funkce aplikace, které byly dále předvedeny testovací skupině.

Implementační část zde byla provedena jako zpětná vazba od testovací skupiny, která poskytla cenné rady a názory na funkčnost a přívětivost aplikace. Na základě těchto poznatků bylo provedeno vyhodnocení, zdali vytvořená práce usnadňuje a ulehčuje porozumění jazyku Java a jestli by studenti byli schopni pomocí této aplikace sestavit nějaký jednoduchý program.

[1] Wireframe neboli drátěný model slouží k náhledu funkčních prvků aplikace.

[2] Jádro aplikace zde disponuje jako řídící centrum všech funkcí.

2 Terminologie

K lepšímu porozumění problematice týkající se této bakalářské práce je třeba rovněž vysvětlit některé základní pojmy. Jelikož tato práce vychází z inspirace modelovacích prostředí, bude sekci UML věnován větší prostor, a to kvůli objasnění kritických funkcí využitých v aplikaci.

2.1 WYSIWYG

Tento anglický akronym *„What you see is what you get“* označuje systém, jehož obsah, který je zobrazen při jeho editování, koresponduje s obsahem při jeho konečném zobrazení. [2]

Specializované editory tudíž disponují uživatelským prostředím, které dopomáhá uživatelům vidět podobný výsledný produkt již při jeho editaci. Tím má uživatel větší představu o tom, jak daný výstup bude vypadat a z čeho se skládá. Nedílnou součástí WYSIWYG editorů se pak stává grafické uživatelské prostředí, což má za význam umístění grafických prvků na obrazovce. [3]

2.2 IDE

Integrované vývojové prostředí [4] [5] je robustní software, který je určený pro vývoj a ladění aplikací v předem určeném programovacím jazyku. Zahrnuje širokou škálu nástrojů - od správy projektů po možnosti zvýrazňování syntaxe. Tento software se skládá z editoru zdrojového kódu, kompilátoru a debuggeru.

2.3 API

Programovací rozhraní je často v informatice označováno jako soubor pravidel a specifikací, které programy využívají ke komunikaci mezi sebou. [6] Toto rozhraní slouží jako prostředník mezi těmito programy a poskytuje užitečné nástroje k jejich manipulaci.

2.4 UML

Existuje několik programovacích jazyků, které umožňují uživateli vytvořit spoustu rozličných programů, ale to není cílem jazyka UML[3]. Tento jazyk slouží pro znázornění, navrhování a specifikaci programových systémů. V UML lze modelovat bez specifikace programovacího

[3] Vychází z anglického akronyma „Unified Modeling Language“

jazyka, operačního systému či sítě, a je tak nezávislý na těchto aspektech. Není to tedy programovací jazyk, avšak má syntaxi a sémantiku[4]. Jelikož tento jazyk podporuje objektově orientovaný přístup, stává se vhodným pro projekty, které využívají ke svému vytvoření objektově orientovaný jazyk.

Modelováním v tomto jazyce dosahujeme grafického návrhu aplikace ještě před jeho kódováním. Navrhuje se, jaké prvky daný projekt bude obsahovat, z jakých částí se bude skládat a kdo se na nich bude podílet. K tomu dopomáhají tzv. diagramy, které znázorňují dané postupy a jejich realizaci při návrhu a jsou vodítkem při vývoji projektu.

UML disponuje dvěma hlavními druhy diagramů. Statické diagramy nebo též také strukturální diagramy popisují neměnnou logickou strukturu programových prvků zobrazením tříd, objektů a datových struktur a vztahy mezi nimi. Dynamické diagramy charakterizují chování entit během spuštění programu tím, že znázorňují jeho průběh nebo způsob jakým se jednotlivé prvky v průběhu mění. [7] V této práci se bude pojednávat o některých těchto diagramech, které souvisejí s návrhem aplikace.

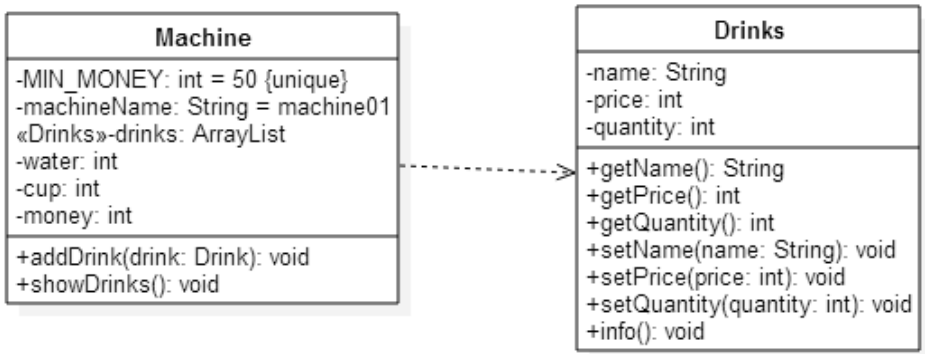
2.4.1 Diagram tříd

Diagram tříd se stává nedílnou součástí objektově orientovaného modelování. Tento diagram se dá použít jak pro konceptuální modelování systematiky aplikace, tak pro přenos modelů do programovacího kódu. Třídy zde představují jednotlivé prvky a interakce, které budou následně naprogramovány.

Diagram tříd je znázorněn jako box, který se skládá ze tří částí, viz níže:

- Horní oddíl nese název třídy. Zobrazuje se tučně, na středu, s velkým počátečním písmenem.
- Prostřední oddíl obsahuje atributy třídy, které jsou zarovnány k levému okraji a píší se s malým počátečním písmenem.
- Dolní oddíl slouží k zapsání operací/metod, které může třída spustit. Jejich zápis je stejný jako u atributů.

[4] Pravidla, jejichž syntaktickým výrazům je přidělen význam [30]



Obr. 2-1 Znázornění diagramu tříd se třemi oddíly

S cílem dále popsat chování systémů jsou tyto diagramy tříd doplňovány o stavové a sekvenční diagramy.

Syntaxe

V UML se atributy a operace zapisují v určitém tvaru, které diagramu dávají význam a jsou posléze snadno převedeny do programovacího kódu.

Atribut

Zápis takovéhoto atributu má tvar:

```
<<stereotyp>> + jmenoAtributu : datovyTyp [*] = hodnota
                                {vlastnosti}
```

- **Stereotyp** uvádí kategorii daných atributů a jedná se o nepovinnou složku.
- **Znaménko** + značí viditelnost atributu, tj. odkud bude přístupný. Další viditelnosti mohou být následující: [8]
  - \+ veřejný modifikátor může být přístupný ze všech tříd,
  - \- soukromý modifikátor je přístupný pouze uvnitř třídy,
  - # chráněný modifikátor bývá dostupný pro vnitřní třídu a její potomky,
  - ~ modifikátor viditelný elementům uvnitř balíčku, ve kterém se nachází vnitřní třída, a balíčkům vnořeným.
- Za modifikátorem přístupu následuje **vlastní název** atributu, který se stává povinným. Napsáním názvu velkými písmeny naznačujeme, že se jedná o konstantu, popřípadě jejím podtržením dáváme najevo statickou vlastnost atributu.

- Datový typ znázorňuje, jakou jednotku vlastnímu názvu určujeme. Oddělují se od sebe dvojtečkou. Může se jednat o primitivní datové typy, či o samotné objekty. Primitivní datové typy se obvykle píší s malým písmenem, kdežto objektové nesou název jejich třídy, tudíž se píší s počátečním velkým písmenem.
- Hodnota v hranatých závorkách udává násobnost atributu. O násobnosti bude ještě pojednáno ve vztazích.
- Atributu můžeme též nastavit jeho implicitní hodnotu při vzniku instance, avšak není to povinné.
- Hodnoty ve složených závorkách umožňují specifikovat další vlastnosti atributů, např. {unique} unikátnost atributu.

Operace

Operace transformují stav svého objektu, čímž dochází k jeho změnám nebo se informují o jeho stavu a následně vrací hodnotu tazateli. Syntaxe operace je téměř podobná syntaxi atributu s několika rozdíly:

```
<<stereotyp>> + nazevOperace(parametr1:datovyTyp1[*],…):
                navratovyTyp {vlastnosti}
```

- Vlastní název operace může být navíc oproti atributu též abstraktní, což se znázorňuje pomocí kurzívy.
- Do kulatých závorek se poté píší parametry a jejich datový typ.
- Na konec operace se zapisuje datový typ, který má operace po vykonání vrátit. Pokud návratový typ není uveden, operace žádnou hodnotu nevrací (void).
- I operace může být opatřena dalšími vlastnostmi, kterými mohou být například výjimky (exceptions).

Vztahy

Na základě komunikace mezi jednotlivými entitami se rozlišují různé vztahy, které plní svou specifickou funkci a zohledňují se v UML diagramu jako takzvané asociace.

Asociace

Vztahy mezi samotnými instancemi[5] popisují právě asociace. Samotná asociace reprezentuje instanční proměnné třídy, které nesou reference na jiné objekty [9]. Například Obr. 2-2 znázorňuje asociaci mezi `Knihou a Stránkou`. Směr šipky napovídá, že `Kniha` nese odkaz na `Stránku`. Název nad šipkou poté označuje název instanční proměnné. Číslo u šipky udává kolik referencí je proměnné přisuzováno, taktéž nazýváno jako Multiplicity (Násobnost).

Obr. 2-2 Znázornění asociace

Agregace

Asociace má spoustu variací a jednou z nich je i agregace. Tento vztah představuje určitou část celku vztahu. K agregaci tedy může dojít, pokud třída zastává funkci kolekce, či kontejneru ostatních tříd, avšak tyto třídy nejsou plně závislé na tomto kontejneru. Tím dochází k tomu, že objekty uvnitř kontejneru nejsou automaticky smazány, kdežto kontejner ano.

Agregace může být pojmenována a mít stejné náležitosti jako asociace. Nicméně agregace se nesmí vztahovat na více než dvě třídy. Graficky se agregace znázorňuje jako prázdný diamant.

Obr. 2-3 Agregace mezi Hřištěm a Divákem

Kompozice

Dalším typem asociace se stává kompozice, což je více specifická forma agregace. Na rozdíl od agregace má tento vztah plnou závislost mezi kontejnerem a jeho instancemi. Pokud je tento kontejner smazán, budou následně smazány i instance v něm obsažené. V UML diagramu tento vztah reprezentuje vyplněný diamant.

[5] Tento vztah též nazýván jako Instance-level relationships

Obr. 2-4 Kompozice mezi Knihou a Stránkou

Násobnost

Syntaxe násobnosti, neboli kolik objektů se v libovolném čase účastní vztahu, může být ve tvaru čísel, rozsahu, či kombinací obou.

Zde jsou povolené formy zápisu:

0	Žádná instance
0…1	Žádná instance, nebo jedna instance
1	Právě jedna instance
0…*	Žádná nebo více instancí
*****	Žádná nebo více instancí
1…*	Jedna nebo více instancí

Generalizace

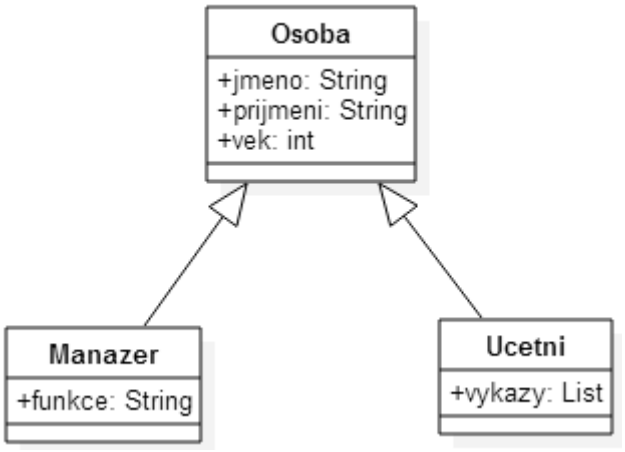
Vztahy na základě tříd[6] jsou rozlišovány několika způsoby; jedním z nich je generalizace. Tento typ vztahu spočívá ve specifikování jedné obecné třídy[7], z níž jsou odvozovány třídy ostatní[8]. Ve výsledku to znamená, že každá instance odvozené třídy je též instancí obecné třídy. V reálném světě to můžeme vyjádřit tak, že člověk je potomkem savců, atd.

V UML diagramu se generalizace znázorňuje jako prázdný trojúhelník směřující k předkovi. Z hlediska programování se tomuto vtahu říká dědičnost, tudíž potomci dědí proměnné, metody, aj. po svém předkovi.

[6] Označovány jako Class-level relationships [33]

[7] Předek neboli Superclass

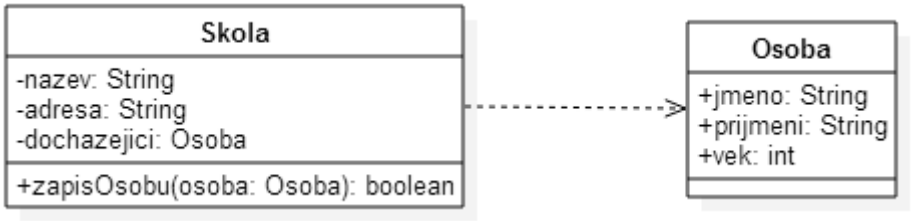
[8] Potomek - též označován jako Super type



Obr. 2-5 Diagram tříd s použitím generalizace

Závislost

Nejvíce využívanou vazbou mezi třídami se v diagramu tříd stává závislost. Jedná se o základní, jednoduchý vztah, který indikuje závislost jedné třídy na druhé. Tento vztah nastává tehdy, když atribut závislé třídy vytváří instanci nezávislé třídy. Vyobrazení tohoto vztahu má tvar přerušované šipky směrem k využívané třídě.



Obr. 2-6 Zobrazení závislosti mezi Školou a Osobou

2.4.2 Diagram objektů

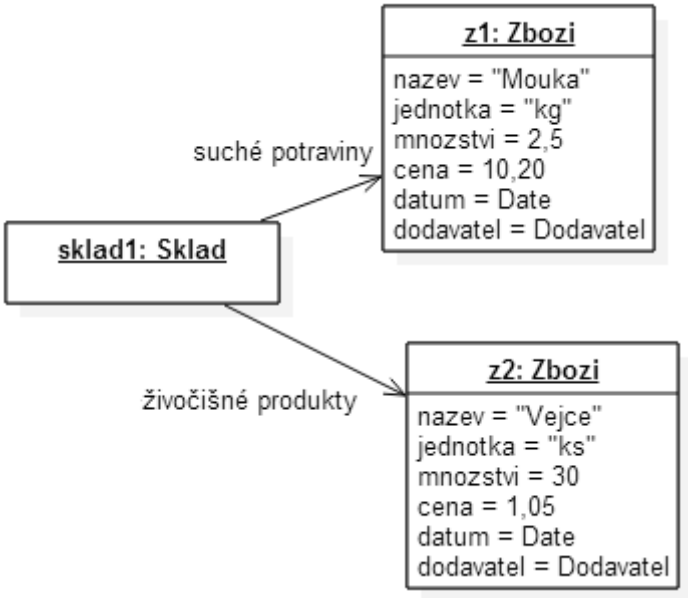
Účelem diagramu objektů je znázornit objekty a jejich relace v určitém čase. Zaměřuje se tedy na určité sady objektů a jejich atributů a spojení mezi nimi. V dřívější verzi UML byl diagram objektů specifikován takto: „*Objektový diagram je graf instancí, včetně objektů a datových hodnot. Tento statický diagram je instancí diagramu tříd; je znázorněn jako snímek podrobného stavu systému v daném časovém okamžiku.*“[9]

[9] Přeloženo z konsorcia OMG [34]

Aktuálnější verze UML 2.5 specifikace doplňuje diagram objektů o způsob zápisu instancí klasifikátorů. [10]

Diagram objektů úzce souvisí s diagramem tříd a mají téměř podobnou notaci. Zatímco diagram tříd zobrazuje třídy neboli souhrn **obecných** atributů a metod, diagram objektů znázorňuje instance těchto tříd a jejich **konkrétní** hodnoty. Tento diagram se často používá ke zvýraznění příkladů jednotlivých prvků diagramu tříd a ověření jeho přesnosti a správnosti.

Graficky se tento diagram znázorňuje podobně jako diagram tříd. Na rozdíl od diagramu tříd se objektový diagram skládá ze dvou oddílů. Horní oddíl nese název objektu a dolní oddíl představuje jednotlivé atributy objektu.



Obr. 2-7 Objektový diagram Skladu

Syntaxe

Samotný zápis hlavičky objektu se poměrně liší od diagramu tříd a vypadá následovně:

`vlastniNazevObjektu : NazevTridy`

- Název objektu slouží k identifikaci objektu, pokud zůstane prázdný, jedná se o anonymní objekt.
- Název třídy slouží k určení, z jaké třídy byl objekt vytvořen.
- Rozlišení, zda se jedná o objekt, se znázorňuje podtržením celé hlavičky objektu.

V definici atributů, též nazývaných sloty, se neuvádí datový typ, přístupnost, atd. Uvádí se pouze název atributu a jeho hodnota. Hodnota atributu může být prázdná, nebo obsahovat i několik hodnot (objekty, pole, kontejnery, …).

Vztahy

Vztahům mezi objekty se zde říká propojení (link). Právě instance asociace dává vzniknout propojení. I vztahy mezi objekty mohou nést nějakou roli, tak jak je tomu na Obr. 2-7, kdy objekt z1 hraje ve skladu objektu sklad1 roli suché potraviny.

3 Analýza

Ke správnému zjištění, co by měla aplikace této práce obsahovat, je analýza zaměřena na ty programy, které byly volně k dostání a se kterými měl autor práce možnost pracovat. Na základě zkušeností byly určeny jejich silné a slabé stránky, které pomohly s návrhem a inspirovaly autora k tvorbě vlastního díla.

Na Internetu je těchto aplikací mnoho, některé přímo podporují reálný vývoj, jiné dopomáhají uživatelům s jejich návrhem. Jejich zaměření a vlastnosti se od sebe liší, tudíž byly roztříděny do několika kategorií.

3.1 Profesionální integrovaná vývojová prostředí

První z kategorií aplikací určených k programování jsou ty, které používají profesionální programátoři a často bývají na vysoké úrovni spolehlivosti. Tato vývojová prostředí v sobě nesou velikou výhodu efektivnosti a usnadnění pro zkušenější uživatele, kteří mají s programováním již nějakou praxi a nabízejí kompletní řešení pro koncové aplikace.

3.1.1 NetBeans

Jedním z profesionálních vývojových prostředí je freeware a open source[10], zvaný NetBeans, od společnosti Oracle. Tento software je jedním z výjimečných integrovaných vývojových prostředí. Dovoluje vyvíjet aplikace v různých programovacích jazycích, včetně Java, HTML5, PHP, C++ aj. Toto vývojové prostředí poskytuje podporu pro kompletní vývojové řešení, od vývoje až po ladění, členění a výsledné vytvoření projektu [11].

Co se funkčnosti a samotného programu týče, disponuje NetBeans několika aspekty, které určují dokonalý nástroj pro snadný a jednoduchý vývoj aplikace.

Vlastnosti a přívětivost programu

NetBeans dokáže doplňovat slova a závorky za uživatele a zvýrazňovat části zdrojového kódu syntakticky a sémanticky. Díky pomocným nástrojům umožňuje snadno refaktorovat [11]

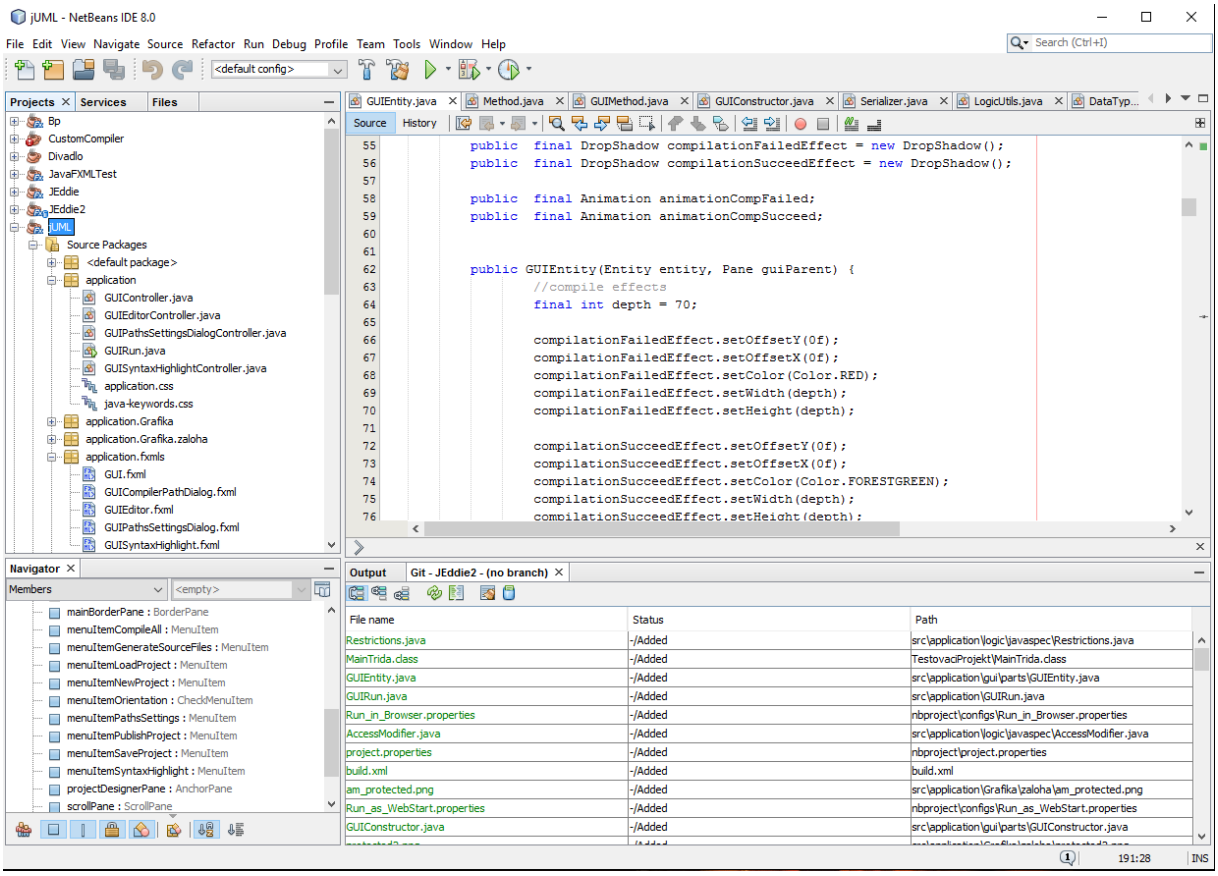
[10] Zdrojově i legálně dostupný počítačový software.

[11] Vylepšení vnitřní struktury kódu bez rizika vnějšího chování.

a poskytuje šablony, či generování kódu. NetBeans přináší také možnost vytváření vlastního GUI[12], díky integrovanému nástroji GUI Builder.

I když se toto IDE pyšní, jak umí dokonale spravovat vytvořené projekty a navigační okna, občas v tomto ohledu ztrácí na přehlednosti a dostane se do problému, kdy začínajícímu uživateli způsobí na obrazovce chaos. Tak zvané uživatelské rozhraní (UI) tedy může zprvu odradit a „zastrašit“ uživatele díky spoustě tlačítek a oken. Přestože lze UI do jisté míry upravit k obrazu svému, riziko zavření některého důležitého okna a následného hledání, kterým tlačítkem se znovu otevírá, je opravdu vysoké. (Obr. 3-1)

NetBeans ve své podstatě dokáže téměř cokoli a je tedy vhodný pro vývoj náročnějších aplikací, avšak jeho robustnost brání začínajícím programátorům se v tomto programu vyznat, natož se naučit programovací jazyk.



Obr. 3-1 Vývojové prostředí NetBeans

[12] Z anglického Graphic User Interface, což znamená grafické uživatelské rozhraní.

3.2 Vývojová prostředí zaměřená na vzdělávání

Nedílnou součástí rozvoje programátorské činnosti bývají aplikace, které nabízejí svým uživatelům přívětivé uživatelské prostředí. Začínající programátor si volí svůj první jazyk na základě dostupnosti a obtížnosti a není tomu jinak ani ve výběru vývojového prostředí.

3.2.1 BlueJ

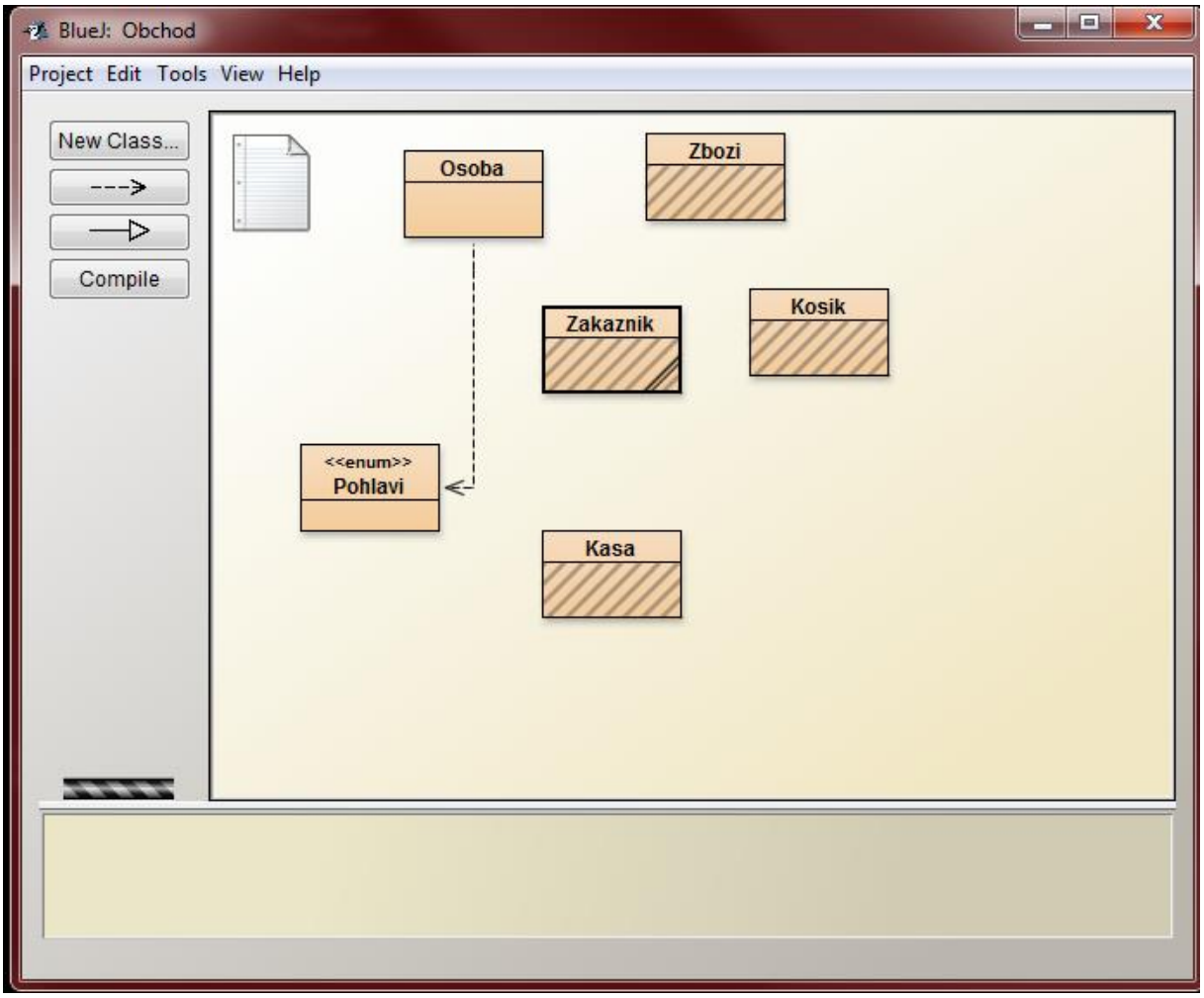
Zajímavým projektem se v pozdějších devadesátých letech stal BlueJ. Michael Kölling, který tento software vytvořil za účelem výuky programovacího jazyka, upoutal pozornost nejen programátorů, ale také pedagogů. Vznikla tudíž jednoduchá aplikace, kterou používají střední i vysoké školy k výuce programovacího jazyku Java po celém světě. (Obr. 3-2)

BlueJ se skládá ze dvou hlavních částí: z pracovní plochy a vytvořených objektů. Pracovní plocha graficky znázorňuje strukturu tříd vytvářeného projektu, objekty mohou být interaktivně vytvořeny a testovány za chodu programu.

Vlastnosti

Jednoduchost a uživatelské rozhraní jsou jednou z výhod, kterou se BlueJ pyšní. Žádné složité uživatelské rozhraní, které by narušovalo pozornost, a nijak zvlášť obtížné ovládání. Po vytvoření nového projektu se uživatel může pustit do návrhu své práce. Pracovní plocha pomáhá k přehlednosti a úpravě daných tříd a připomíná tak UML diagram.

Další předností této aplikace je především interakce s objekty. Můžou se dynamicky prozkoumat a testovat, tudíž jsou vidět jejich hodnoty, konstruktory a metody, které mohou být dále vyvolávány. Na základě této vlastnosti se uživatel dostane hlouběji do problematiky programování [12].



Obr. 3-2 Vývojové prostředí BlueJ

Samotný editor kódu je velmi přehledný, každý blok je zvýrazněn, taktéž syntaxe a některé hodnoty jsou barevně odlišeny. Součástí editoru je též integrovaný kompilátor, který třídy zkontroluje a zkompiluje.

Nicméně i tento užitečný nástroj má své neduhy. Při projektech, které mají více tříd, začíná být jednoduchost rozhraní na obtíž. Každá třída otevře nové okno editoru a tím způsobí zmatek na obrazovce, hlavně když těchto tříd je otevřeno více. Také jakákoli změna v dané třídě má za následek otevření okna s onou třídou.

3.3 Modelovací prostředí

Nedílnou součástí vývoje aplikace se stává návrh jejího vzoru. Pokud programátora napadne idea projektu, nejlepší cestou je tuto myšlenku rozvést a vytvořit si návrh tohoto projektu. S tímto dopomáhají nejrůznější nástroje, jako jsou myšlenkové mapy, wireframe, či UML

diagramy. Pro účely této bakalářské práce byly analyzovány poslední zmíněné, především jejich prostředí.

Tyto programy pro tvorbu UML pomáhají uživatelům za pomocí nejrůznějších diagramů vytvářet logickou strukturu jejich díla. (viz *kapitola 2.4 UML*)

3.3.1 StarUML

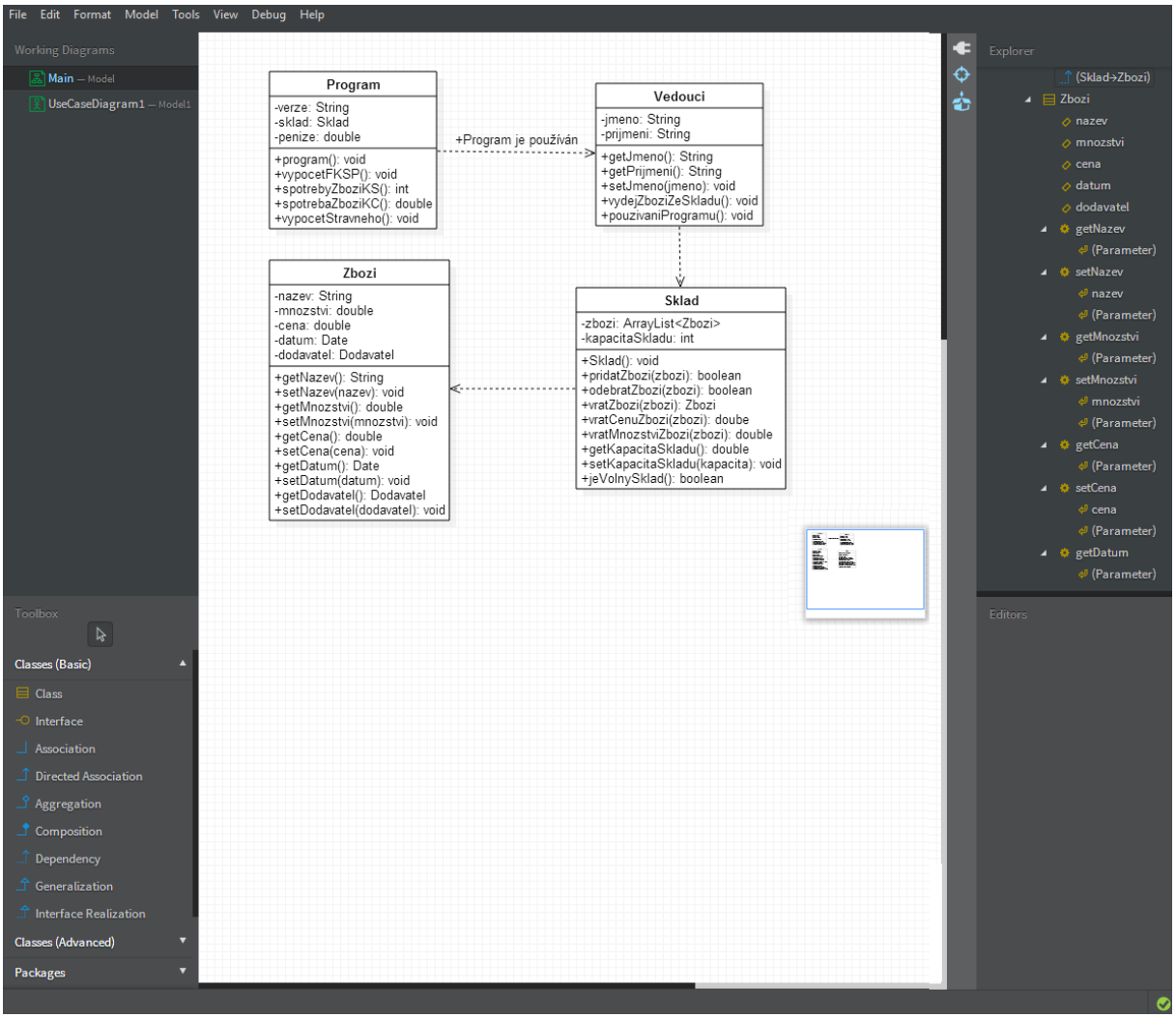
Nejpopulárnějším nástrojem pro tvorbu UML diagramů se stává projekt StarUML, který nabízí celkem až 11 druhů diagramů. Disponuje širokou škálou nástrojů, jak projekt spolehlivě navrhnout, na funkcionalitě tudíž nešetří a stává se profesionálním návrhovým prostředím [13].

Vlastnosti a přívětivost programu

I přes svoji robustnost se s tímto projektem uživatel nemusí bát zacházet, protože jeho uživatelské prostředí je velice přívětivé a jednoduché. Uživatelské rozhraní je chytře navrženo tak, že se uživatel soustředí jen na pracovní plochu a nevyrušují ho přebytečná tlačítka, či špatně umístěný panel nástrojů. Onen panel nástrojů je velmi jednoduchý na ovládání, stačí jen kliknout na zvolený objekt a poté se při opětovném kliknutí na pracovní plochu vytvoří jeho grafická reprezentace.

Upravovat vlastnosti jednotlivých prvků lze dvěma způsoby. Prvním z nich je přidávání atributů přes kontextovou nabídku při stisknutí pravého tlačítka, což šetří prostor pro pracovní plochu. Druhým způsobem je úprava vlastností v okně editoru, který se nachází v pravé dolní části programu. (Obr. 3-3)

Tento program dává svým uživatelům do rukou i velmi užitečný nástroj, tím je možnost importování různých rozšíření. Například takové vygenerování Java kódu z UML modelu může být pro někoho velmi cenným přínosem a ušetří mu spoustu času. Nicméně pro další editaci onoho kódu již musíme zvolit jiná prostředí.



Obr. 3-3 Modelovací prostředí StarUML

3.4 Shrnutí analýzy

Každé z výše popsaných prostředí skrývá své přednosti i slabé stránky, které jsou pro vývoj aplikace nezbytné.

Integrovaná vývojová prostředí zaskočí svým robustním a ne úplně jednoduchým uživatelským rozhraním, ale plní svoji funkci dokonale a soustředí se zcela na programování kódu, který též pomáhají prostřednictvím užitečných nástrojů jistým způsobem zjednodušit.

Přednost vzdělávacích vývojových prostředí spočívá v jejich jednoduchosti na ovládání a do jisté míry i náhledu do problematiky jazyka. Dokáží uživateli osvětlit jisté principy a poukázat na chyby, které v programování vznikají. Nicméně při větších projektech nastává problém a jednoduché uživatelské rozhraní se stává přítěží.

Na druhou stranu UML software připraví uživatele na opomíjenou součást programování, a tím je návrh. Díky propracovanému návrhu, který tato prostředí umí zvládnout dokonale, se programátor může odrazit a připravit se na samotný vývoj aplikace. Co se však týče samotného psaní kódu, nezbývá než tento software opustit a vyhledat výše zmíněná prostředí, či jiné programovací možnosti.

Tím se dostáváme k samotným kritériím, která by měla splňovat aplikace této bakalářské práce. Jsou to:

- jednoduché a přívětivé GUI,
- schopnost návrhu projektu i psaní kódu,
- náhled do problematiky jazyka (sémantika psaní kódu, syntaxe, atd.).

4 Návrh

Z analýzy vyplynulo několik aspektů, které by měla aplikace této práce obsahovat, a zároveň splňovat stanovené cíle práce. Sám o sobě má pojem WYSIWYG spoustu významů, se kterými se dá velmi dobře pracovat. Nedílnou součástí takovéhoto editoru je právě jeho návrh, aneb jaký koncept zde bude zvolen, jaké funkce a vlastnosti bude obsahovat a v neposlední řadě v jakém jazyce se bude aplikace programovat.

S výběrem programovacího jazyka nebyl žádný problém a po zkušenostech s jazykem Java to byla jasná volba. Shodou okolností i samotná aplikace má být pomůckou k objasnění základních principů a vlastností tohoto jazyka, takže i programování v něm dovoluje zamyslet se nad úskalími a přínosy jazyka Java. Navíc samotná Java je jazyk velice rozšířený a disponuje mnoha knihovnami a technologiemi, které budou pro vývoj aplikace velice důležité.

Následující kapitoly popíší autorovu představu o vlastní aplikaci a postup při sestavování jejího návrhu.

4.1 První podnět

Když se řekne WYSIWYG editor, zřejmě se spoustě lidí vybaví některé profesionální modelovací prostředí pro vývoj webů, jako je například Dreamweaver od společnosti Adobe. V tomto prostředí pak může uživatel graficky navrhovat design svého webu a popřípadě upravovat jeho zdrojový kód.

Na rozdíl od webdesignu, kde jsou jednotlivé prvky vidět na webové stránce, se při programování jednoduchých projektů s tímto nesetkáme. K tomu nám však slouží jiné pomůcky. Pokud chce programátor graficky znázornit svůj projekt, sestaví si jeho návrh v jednom z modelovacích prostředí. Tato prostředí poté dovolují programátorovi vidět celkovou strukturu jeho projektu. Z analýzy však vyplynulo, že tato prostředí neumožňují uživateli zapisovat kód, či kompilovat jednotlivé entity.

Zde vznikla myšlenka vytvořit podobný koncept UML diagramů, respektive diagramu tříd, a zakomponovat ho do prostředí Javy. Tím by vzniklo unikátní prostředí pro modelování projektů s implementací editoru kódu, a to vše pro podporu jazyka Java.

Tento koncept dovoluje soustředit se na návrh jednoduchého projektu a na editaci a vytváření struktury jednotlivých tříd. Následně bude aplikace generovat ukázkový zdrojový kód, který

bude z hlediska syntaxe a sémantiky jazyka Java správný a bude případně dovolovat uživateli vytvářet vlastní kód a upozorňovat na možné chyby v syntaxi.

4.2 Vytváření návrhu aplikace

Základní myšlenka aplikace již byla vyřčena, tudíž prvním milníkem ve vývoji je skutečný návrh této aplikace. Tento návrh spočívá v několika fázích.

V první řadě je třeba představit si, jak by měla výsledná aplikace zhruba vypadat. V dalším kroku by se mělo zmínit, jak se s ní bude zacházet a jakými funkcemi a vlastnostmi bude disponovat. Poté následuje příprava na samotný vývoj aplikace za pomoci modelovacích nástrojů a schémat.

V této práci budou zakomponovány diagramy, nejrůznější schémata a obrázky, které byly pořizovány v průběhu návrhu i vývoje pro znázornění myšlenek a principů projektu.

4.3 Grafická struktura aplikace

Grafické schéma, jak by měla aplikace vypadat, se neobejde bez ovládacích prvků, které budou uživateli k dispozici. Chceme-li vytvořit opravdu minimalistické grafické prostředí, které nebude zatěžovat uživatele zbytečnými prvky na obrazovce, zvolíme k tomuto účelu odpovídající rozvržení.

Jak již bylo řečeno, aplikace se bude podobat UML diagramu tříd, který přináší vlastnosti WYSIWYG editoru a umožní uživateli navrhnout si v modifikovaném prostředí svůj vlastní Java projekt.

4.3.1 Layout

Správné rozložení jednotlivých prvků na scéně[13] je jeden ze základních aspektů vývojových prostředí. Na rozdíl od layoutů webových stránek [14] je rozvržení u konečných aplikací poněkud složité a využívá se zde i vícero layoutů.

Profesionální IDE prostředí měla právě s tímto velký problém (viz *kapitola 3.1 Profesionální integrovaná vývojová prostředí*) a nedokázala tak maximalizovat pracovní prostor. Ideálním

[13] představující okno aplikace

řešením by tedy bylo zaměřit se na modelovací plátno a minimalizovat tak počet sekcí[14] layoutu. Předpokladem by také měla být znalost rozsahu uživatelova projektu, tudíž se pracovní plocha musí této změně přizpůsobovat.

Výsledkem tedy musí být minimalistický layout s co nejmenším počtem oken a s interaktivní pracovní plochou, která bude uchovávat modelované objekty.

4.3.2 Ovládací prvky

Na základě stanovení množství ovládacích prvků aplikace se odvíjí i její možnosti a funkcionalita. Jelikož byla inspirace čerpána z UML diagramu tříd, hlavním ovládacím prvkem zde bude samotná třída prezentovaná jako entita.

Následné ovládání bude prostřednictvím grafické entity, kde bude možné nastavovat a upravovat její vlastnosti. Čím se však bude tato aplikace lišit od standardních modelovacích prostředí, je schopnost vkládání či editace zdrojového kódu jednotlivých částí entity.

Entity

Tvoří hlavní ovládací prvek aplikace, se kterým je možné manipulovat na modelovacím plátně.

Vztahy

Mezi jednotlivými entitami vznikají vztahy na základě jejich užití. Prezentovány jsou jako různé šipky podle jejich typu.

Kontextové menu

K širšímu využití vytvářených entit bude rozumné vytvořit kontextové menu, které nebude zabírat místo. Pomocí tohoto menu by se vyloučilo vytváření panelu nástrojů a zjednodušilo ovládání aplikace [15].

[14] Myšleno počtem sloupců či vnitřních oken aplikace.

Editor zdrojového kódů

Je nutná implementace editoru kódu entity za účelem okamžité úpravy či vkládání zdrojového kódu jazyka Java s následným přidáváním jazykových struktur pomocí kontextového menu.

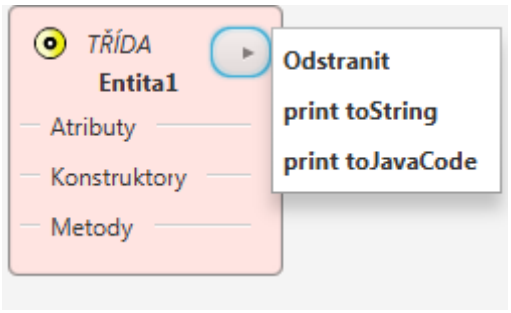
4.3.3 Grafická reprezentace entity

Tak jako diagram tříd bude i reprezentace entity této aplikace představovat čtverec rozdělený do několika částí. Cílem však je naučit uživatele této aplikace základní principy jazyka Java, a tudíž tomu musíme přizpůsobit jak vzhled, tak i funkčnost entity.

Standardně je třída v UML diagramu rozdělena do tří oddílů[15] (viz Diagram tříd). Java se však skládá ze tří hlavních elementů: atributy, konstruktory a metody [16]. Na základě těchto elementů a typu třídy může vzniknout entita s několika oddíly, které mohou být následující:

- název a typ třídy,
- seznam atributů,
- seznam konstruktorů,
- seznam metod,
- výčtové typy.[16]

Tyto oddíly by poté nesly jednotlivé elementy vytvořené uživatelem. Vyobrazení těchto prvků by bylo stejné jako syntaxe UML diagramu tříd.



Obr. 4-1 Případná ukázka entity

[15] V diagramu tříd se konstruktor zapisuje jako operace.

[16] Jedná-li se o třídu Enumeration.

Modifikátor přístupu

Konvence jazyka Javy rozlišuje několik typů modifikátorů přístupů, které slouží za účelem přístupnosti a viditelnosti elementu jiným objektům. V UML diagramu jsou znázorněny jako speciální znaky (viz *str. 15*).

Jedním ze způsobů jak odlišit modifikátor přístupu v této aplikaci bude navržení vlastní grafické reprezentace. Zajímavou myšlenkou může být, jak už i samotná definice napovídá, tvar oka, jehož tvar či barva se mění v závislosti na jeho typu.

Obr. 4-2 Ukázka modifikátorů přístupů

Atributy

V tomto případě budou atributy představovat instanční proměnné dané entity. Grafický zápis bude totožný se zápisem diagramu tříd. To dovoluje uživateli předběžně se seznámit se zápisem atributů v UML editorech.

Jako tradičně nesmí atributu chybět modifikátor přístupu vlastní jméno a datový typ, tím proběhne základní deklarace instanční proměnné. Případným doplněním hodnoty atributu se docílí samotná inicializace [17].

Takto vyplněný atribut bude posléze automaticky vygenerován do syntaxe jazyka Java a může být prohlížen v editoru kódu.

Konstruktory a metody

Stejně jako grafický zápis atributů i zápis konstruktorů a metod bude podobný. Jelikož konstruktor je sám o sobě metoda, pro lepší přehlednost a znázornění jednoho z hlavních elementů Javy byl zařazen do samostatného oddílu. Konstruktor též plní funkci inicializátora [18] a vytváří instance dané třídy, proto by měl být jeho název shodný s názvem třídy.

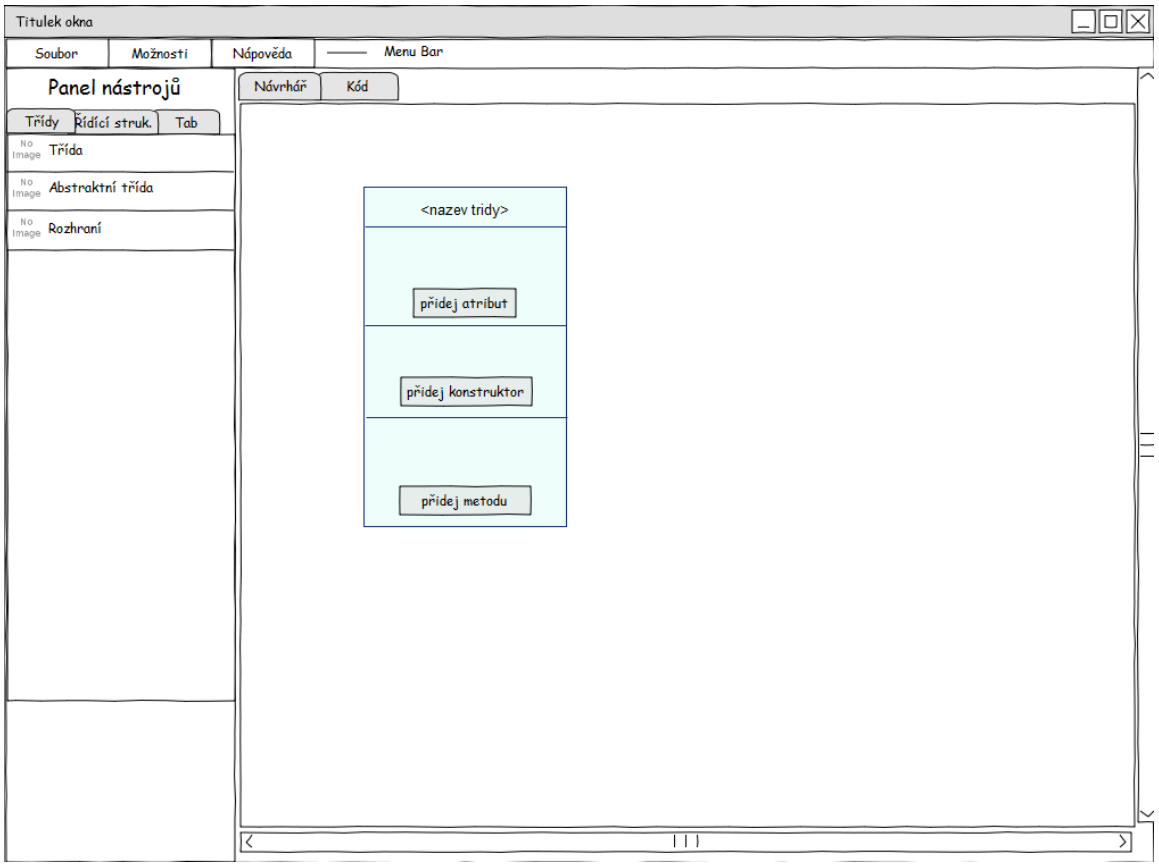
Co by měly mít konstruktor a metoda společného, je pole parametrů a možnost vkládání a upravování tělíčka metod a konstruktorů. Co se týká parametrů, měla by aplikace umožňovat jejich přidávání a odebírání s následným zobrazením vlastního názvu nebo datového typu parametru.

Editor kódu bude tedy sloužit pro pokročilejší editaci entity, a to zejména při navrhování vlastních metod či rozšíření konstruktoru o progresivní kód. Více o editoru kódu bude popsáno v pozdějších kapitolách.

V neposlední řadě se v jazyce Java rozeznávají tzv. *Getters* a *Setters* [19]. Ty určují, zda se jedná o metodu s návratovým typem či nikoli. Tyto metody a jejich definice návratového typu musí být též určeny v grafické reprezentaci aplikace.

4.3.4 Wireframe

Názornou ukázku, jak by mohla taková aplikace vypadat, lze vytvořit pomocí wireframu[17]. Základní aspekty byly již zmíněny, tudíž by měl následující obrázek znázornit hrubý náčrt aplikace.



Obr. 4-3 Počáteční návrh WYSIWYG editoru

Jedná se o počáteční náčrt a výsledek se může kdykoli v průběhu vývoje změnit, nicméně grafická část aplikace byla dostatečně popsána a následuje její logická část.

[17] Drátěný model znázorňuje projektovou dokumentaci aplikace [35].

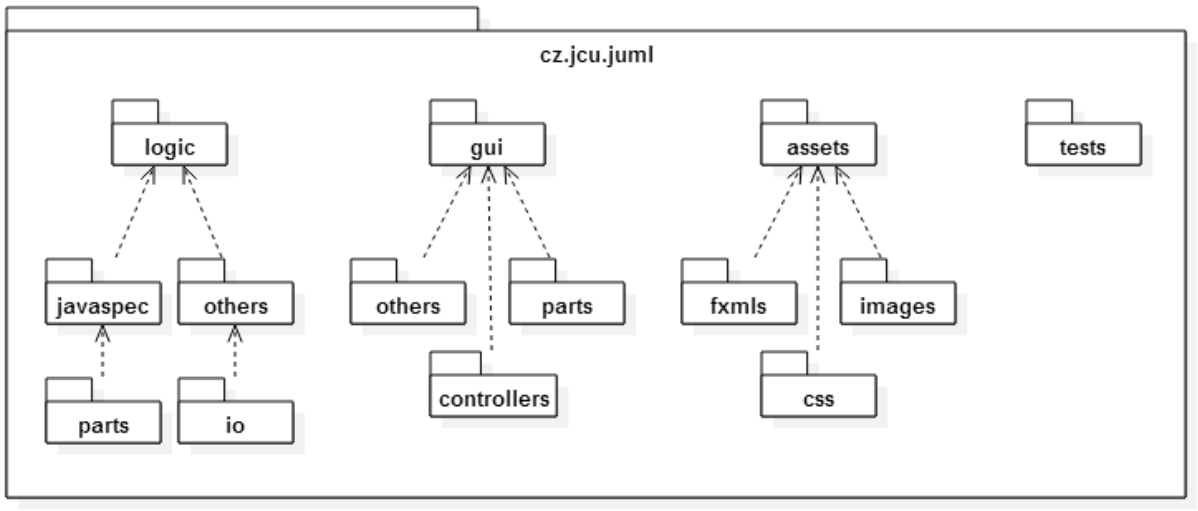
4.4 Logická struktura aplikace

Jelikož se předpokládá, že aplikace bude většího rozsahu z hlediska počtu entit a dalších souborů, bude třeba tyto soubory třídit do nějaké smysluplné struktury. Stejně jako v jiných pokročilých aplikací a též dle standardu vývoje [20] se budou části aplikace strukturovat do různých balíčků, jež v jazyce Java rozeznáváme pod pojmem *„packages“*.

V hlavní úrovni kořenového balíčku budou entity zodpovědné za spuštění aplikace. Další balíček definuje logické jádro softwaru, což reprezentuje stav otevřeného projektu s možností samotné aplikace. Následně se na tomto jádru vybuduje grafické uživatelské rozhraní, jež poslouží uživateli k jednoduchému zacházení v tomto prostředí.

Druhá část pak staví na této logické části front-end [18] aplikace za využití grafického uživatelského rozhraní JavaFX. Při samotném návrhu bude využit i nástroj Scene Builder, jehož výstupem jsou soubory definující strukturu grafického uživatelského rozhraní ve specifickém formátu FXML. Tyto FXML soubory načítá aplikace za běhu, tudíž budou součástí struktury aplikace ve svém vlastním balíčku FXMLs.

Jednou z výsad této aplikace bude využití kaskádových stylů, nicméně širší spektrum grafického rozhraní se zřejmě neobejde bez vlastních rastrových obrázků, které budou muset být implementovány. Z hlediska metody „rozděl a panuj“ budou tyto obrázky začleněny do dalšího odděleného balíčku.



Obr. 4-4 Diagram balíčků strukturovaného projektu

[18] Výraz spojený se zajímavou grafickou podobou aplikace, podobný význam má pojem look and feel [31] [32].

4.5 Ukládání a otevírání projektu

Součástí každého dobrého programu je schopnost uložit svůj stav do externího souboru a v případě potřeby provést opačný proces k obnovení předchozího stavu. V rámci výukových programů by se dalo říci, že jde o funkcionalitu nutnou k zajištění úspěchu. Nejde pouze o to, že si student může uložit svůj rozdělaný projekt, ale i sám učitel si může připravit příklady dopředu a na jejich základě obohatit výuku. Proto bude aplikace podporovat i tyto funkce.

Jednou z možností, jak tuto funkcionalitu implementovat, je navrhnout vlastní pořadí atributů a způsob jejich oddělování v cílovém souboru, podle nichž se pak při načítání stavu bude postupovat v opačném pořadí. Tento způsob vyžaduje mnoho programování a testování a není přívětivý k rozšiřování aplikace, kdy bude nutné zanést všechny změny do těchto algoritmů. Lepší řešení ukládání a otevírání projektu lze dosáhnout implementací serializace a deserializace.

Serializace je proces, při němž se objekty držené v paměti počítače převedou do jednodušší formy (struktury), kterou je možné uchovat v souboru na pozdější dobu. Deserializace je proces opačný, při něm dochází k obnovení (respektive znovuvytvoření) stejného počtu objektů s identickými vazbami a totožným stavem.

Pro ukládání a otevírání projektu nemá smysl implementovat vlastní serializační a deserializační algoritmy, které jsou náročné na návrh, vyžadují stanovení pořadí atributů a způsob jejich oddělování, když jazyk Java zavádí nativní podporu serializace.

Třídy objektů, jež máme v úmyslu serializovat, musí implementovat rozhraní *java.io.Serializable* [21]. Poté stačí pracovat s výstupními proudy, typu *File* a *Object*, které převedou objekty do binární formy a tu následně uloží do zvoleného souboru.

Při využívání knihoven třetích stran či některých okrajových oblastí JDK, nemusí být u některých tříd podpora serializace. Použití těchto datových typů v našich třídách je učiní též neserializovatelnými. Řešením je vybavit atributy neserializovatelných datových typů klíčovým slovem *transient*, které způsobí, že takto označené objekty budou při serializaci ignorovány. Při vývoji aplikace se můžeme s tímto problémem setkat, budou-li použity entity typu *bindable properties*, kterým chybí podpora serializace [22].

Vynecháním těchto objektů ze serializačního procesu však není možné projekt uložit a zároveň otevřít kompletně. Proto je nutné tyto objekty zahrnout do serializace ‚ručně' v algoritmu za použití již zmíněných výstupních proudů a serializovat všechny atributy.

4.6 Převod entit do zdrojového kódu

Uživatel by měl mít možnost doplnit zvolené části modelovaných entit o programový kód jazyka Java. Jde například o konstruktory a metody tříd. Z toho důvodu musí být jednotlivým částem jádra aplikace definujících entity, jejich atributy a metody, doprogramována podpora pro převod (reprezentaci jejich stavu) do textové podoby odpovídající klíčovým slovům a syntaxi jazyka Java.

Výhodou implementace této funkcionality je nejenom konečné sestavení například tříd, které bude možné uložit v textové podobě do souboru a poté zkompilovat, ale také to, že definice atributů a hlavičky metod entit dokáže aplikace do programového kódu automaticky vygenerovat na základě modelované struktury entity, již uživatel nastaví přes GUI.

Třídy umístěné v logice aplikace reprezentující jednotlivé členy entit (atributy, konstruktory, metody) budou implementovat pomocné rozhraní *Codeable* obsahující metodu *toJavaCode()* s návratovým typem *String*. Třída toto rozhraní poté implementuje a tím dojde k převodu určených částí do programovacího kódu.

4.7 Kompilace entit a projektu

Díky převodu entit do zdrojového kódu jazyka Java bude umožněna zásadní část této aplikace a tj. kompilace jednotlivých entit [23]. To otevírá široké možnosti využití jak s daným projektem nakládat. V první řadě bude možné ověřit si, jestli modelované entity a jejich struktura odpovídají syntaxi Javy. Tím vzniknou dvě situace. Za prvé bude nutné vypisovat případné chyby v zápisu syntaxe. Za druhé pokud kompilace proběhne v pořádku, měla by se tato informace znázornit uživateli.

4.7.1 Základní využití kompilace

S kompilací zdrojového kódu v Javě pomáhá knihovna *JavaCompiler*, která se stará o přeložení zdrojových souborů do formátu *.class*. Pokud se kompilace z nějakého důvodu nepovede, vrátí proud chyb, který posléze uloží do souboru. Toho lze využít pro zobrazení chyb za běhu programu, a tím tak dát uživateli vědět o případných chybách přímo v aplikaci ve formě log[19] okna.

[19] Jedná se o zobrazení souboru záznamů s informacemi o průběhu určité činnosti.

Graficky lze jednoduše znázornit zkompilované entity například změnou barvy či jinými efekty. Tím se vyřeší problémy způsobené kompilací a názorně se ukáže uživateli, kde přesně udělal chybu. V tomto případě by kompilace pomohla zajistit, aby uživatel vytvářel syntakticky správný model a popřípadě se seznámil s chybami, kterých se v zápisu tělíček dopustil.

Tuto funkcionalitu by měla aplikace této práce implementovat, aby bylo možné uživateli ukázat, jak se dané entity přetvářejí na zkompilované soubory a jak se v jazyku Java píše syntakticky správný kód.

4.7.2 Rozšířené funkce kompilace

Další možností, kterou přináší zkompilovaný projekt, je schopnost jeho spuštění přes metodu main. Pokud by se jednalo o pokročilejší vytváření projektu, kde by výsledkem byl samostatný spustitelný program, musela by tato aplikace implementovat tuto metodu. Teoreticky by se dala určitá entita označit jako ‚hlavní', která by tuto metodu implementovala v její nejjednodušší formě. Tím by se docílilo možnosti spustit projekt například v samostatném okně či příkazovém řádku.

S tím se pojí i další možnost, a tou je vygenerování samostatně spustitelného souboru, který se váže na metodu main a dovoluje uživateli spouštět jeho vytvořený program nezávisle na prostředí aplikace. K tomuto účelu by opět posloužila knihovna java.util.jar, která je schopna zkomprimovat zdrojové soubory do samostatného spustitelného souboru s koncovkou JAR[20].

Tyto funkce jsou pokročilé a vyžadují poměrně velkou míru adaptace výsledného projektu, mohou však sloužit pro případné rozšíření aplikace, kterou obohatí o další specifické funkce.

4.7.3 Požadavky na kompilátor

Jelikož se tato aplikace snaží o zkombinování modelovacího prostředí s vývojovým prostředím, měl by i samotný uživatel mít určité podpůrné nástroje pro interaktivní technologie, ať už se jedná o Adobe Flash Player, Java SE Runtime Enviroment, Net Framework, Microsoft Visual C++ RE, aj., které zastávají spouštěcí prostředí programů jednotlivých organizací. K účelům vytváření vlastních aplikací poté slouží jejich vývojová verze se všemi knihovnami, balíčky a API.

[20] Formát JAR představuje specifický souborový formát jazyka Java, který je založen na archivačním formátu ZIP [36].

Pokud tedy dojde k začlenění kompilátoru do prostředí aplikace, bude od uživatele vyžadováno nainstalování podpory vývojářského prostředí pro jazyk Java též pod zkratkou JDK[21]. Jak již bylo zmíněno v předchozích podkapitolách, tyto zdrojové soubory obsahují knihovny potřebné ke kompilaci projektu.

4.7.4 Nastavení cest nového projektu a kompilátoru

Při prvním spuštění jakéhokoli IDE softwaru je zapotřebí nastavit několik náležitostí. Povětšinou to bývá zvolení adresáře, kde se budou ukládat jednotlivé projekty. V nejkrajnějším případě pak bývá importování nejrůznějších balíčků a knihoven.

Tato aplikace musí uživateli sdělit, kam chce své projekty ukládat a jestli je třeba načíst nějaké jiné soubory. Z předchozí kapitoly je zřejmé, že bude nutné naimportovat vývojové prostředí jazyka Java. Pokud se uživatel rozhodne nenaimportovat zvolené soubory, je nutné mu zdůraznit, že nebude možné využívat funkci kompilace.

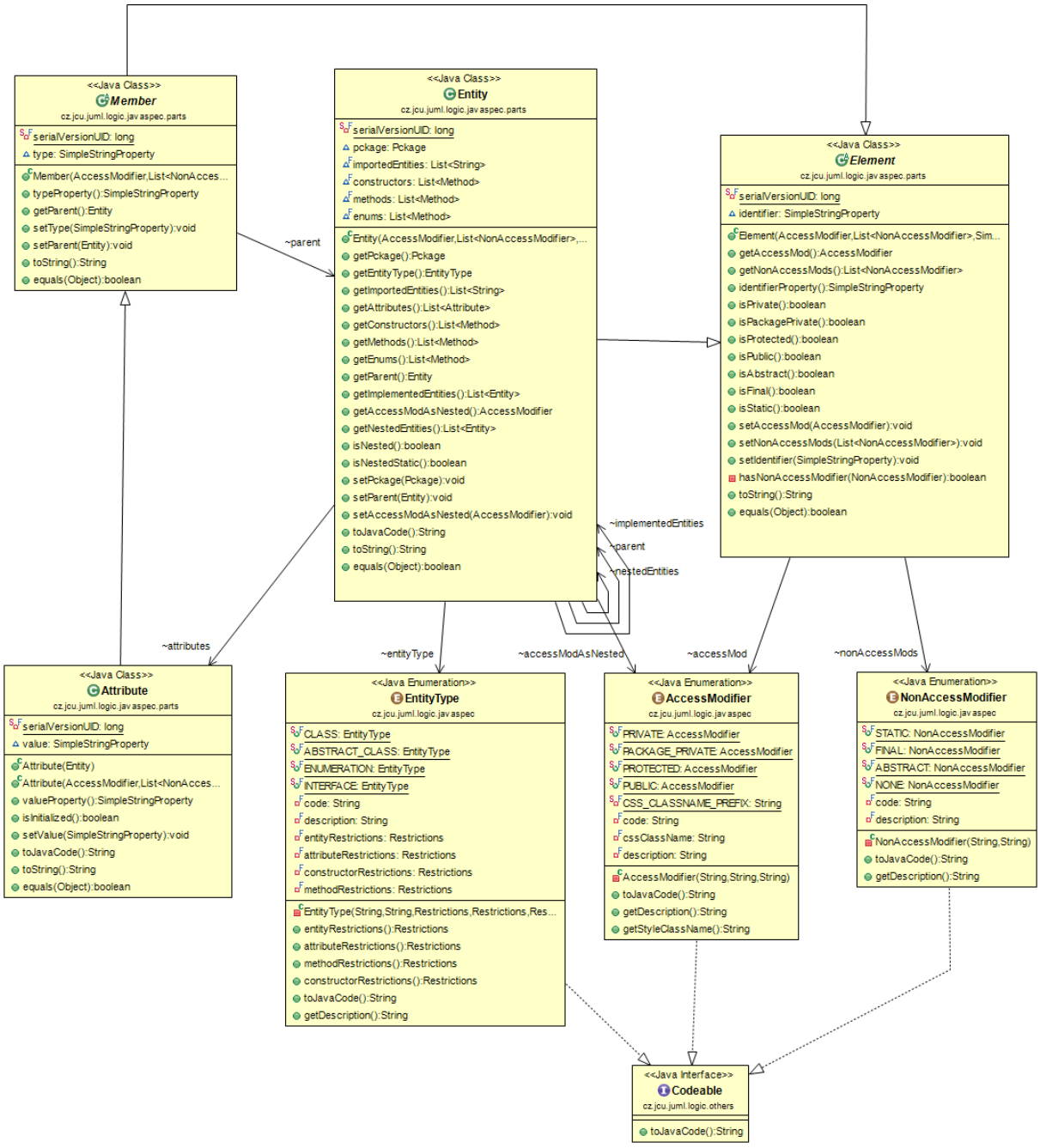
Při zakládání nového projektu by měla být aplikace schopna stávající projekt uložit a vytvořit projekt nový. Možností by mohlo být i automatické ukládání. Princip ukládání a načítání projektu byl již uveden v kapitole Ukládání a otevírání projektu.

4.8 Příprava na vývoj aplikace

Nyní, když byly stanoveny všechny náležitosti potřebné k sestrojení vlastní aplikace, následovalo sestavení podrobného plánu vývoje aplikace. Finální fází při návrhu bylo tedy vytvořit diagram tříd, který by odděloval danou logickou část aplikace od její části grafické.

Z toho vyplývá, že nezávisle na grafické části aplikace zůstane její jádro přístupné k případným modifikacím a její funkcionalita bude zachována. Došlo tedy k vytvoření základní struktury logického jádra, na kterém se bude aplikace vyvíjet.

[21] Java Development Kit je využívána vývojáři pro programování v jazyce Java. Obsahuje knihovny a nástroje k samotnému vývoji koncové aplikace v jazyce Java.



Obr. 4-5 Diagram základních tříd aplikace

5 Vývoj

Ve fázi vývoje probíhalo samotné vytvoření aplikace na základě zmíněných návrhů a postupů. Jednotlivé složky aplikace byly vytvářeny do jistých milníků, které byly zároveň testovány. Tyto revize byly zaslány testovací skupině, která poskytovala velmi vzácnou zpětnou vazbu a která odhalila případné nedostatky aplikace.

Nedílnou součástí vývoje byly též rozličné technologie, které byly v průběhu práce využity. V následujících kapitolách budou popsány jednotlivé milníky a technologie, které byly do aplikace zahrnuty.

5.1 Stěžejní prostředky ve vývoji

Ve vývoji této aplikace byly použity některé nové technologie, kterými Java disponuje a externí nástroje pro efektivitu vývoje, a proto jim bude věnován určitý prostor.

5.1.1 Technologie JavaFX

Velkou roli ve vývoji této aplikace hrála právě technologie JavaFX, která postupně ovládá vývoj mobilních, webových a desktopových aplikací v programovacím jazyce Java. Ve zkratce lze technologie JavaFX popsat jako sadu grafických a mediálních balíčků, které dovolují vývojářům navrhovat a vytvářet bohaté klientské aplikace. Tyto aplikace pak mohou být spuštěny na nejrůznějších platformách [24].

Klíčových vlastností má tato technologie nespočet. V aplikaci se zejména projevují tyto vlastnosti:

Java API

Důležitá součást JavaFX jsou pro vývojáře všechny dostupné třídy, knihovny a balíčky i s podrobnou dokumentací, které aplikace hojně využívá.

FXML a Scene Builder

V návrhu aplikace bylo již zmíněno, že tato technologie umožňuje vytvářet interaktivní uživatelské rozhraní za pomoci modelovacího nástroje Scene Builder [25]. Pomocí tohoto nástroje byly posléze vytvářeny jednotlivé grafické uživatelské části aplikace. Tyto prvky jsou převáděny do souborového formátu *fxml,* který je rozšířenou FX verzí značkovacího formátu xml. V pozdějších kapitolách bude tomuto nástroji věnován větší prostor.

Ovládací prvky UI a CSS

Kde modelování ve Scene Builderu nestačilo a některé části uživatelského rozhraní bylo nutné dodělat ručně, pomohly opět knihovny JavaFX, které disponují základními UI prvky, které známe například z knihoven Swing. Jelikož tato technologie je též určena pro vývoj webových aplikací, disponuje možností přizpůsobovat si komponenty pomocí CSS stylů, které byly též zahrnuty do této aplikace.

5.1.2 Externí knihovny

Pro některé účely spojené s funkcionalitou aplikace musely být použity externí knihovny, které zajistily efektivnost řešení problémů dané situace. Místem, kde se setkávají programátoři z nejrůznějších oborů a dělí se o své výtvory s ostatními, se stává internetový portál GitHub.

Na tomto serveru jsem nalezl velmi talentovaného autora Tomáše Mikulu, který se zabývá vývojem knihoven právě pro prostředí JavaFX. Tím se mi naskytla příležitost použít některé z jeho knihoven v mé vlastní aplikaci. Jedná se hlavně o tyto knihovny[22]:

- ReactFX verze 2.0-M4U1,
- RichTextFX verze 0.6.1,
- Flowless verze 0.4.7,
- ControlsFX verze 8.40.1 [26],
- UndoFX verze 1.2.

Čím knihovny disponují a jak byly v projektu použity, bude popsáno v pozdějších kapitolách. Tímto bych chtěl též poděkovat autorovi za vývoj těchto knihoven, které byly velkým přínosem při vzniku této aplikace.

5.2 Logická část aplikace

Logická část aplikace se soustředí na reprezentaci struktury jazyka Java. Tím se myslí, z čeho se všeobecně tento jazyk skládá a jakým způsobem se zapisuje. Aplikace tedy bude uživatele

[22] Všechny zmíněné knihovny jsou dostupné v repositáři autora pod licencí GNU General Public License (GPL) [37].

provádět tímto jazykem v jednoduché grafické formě. Tyto elementy bylo nutné vytvořit jako první a sestavit tak jádro aplikace.

5.2.1 Jádro aplikace

Rozložením jednotlivých elementů programovacího jazyka vzniknou určité kategorie, které byly v jádru aplikace roztříděny do specifických tříd.

Základním elementem každého programovacího jazyka se stává samotná třída, která v této aplikaci disponuje pod názvem *Entita*. Ta si uchovává informace o svých dílčích částech, jimiž jsou atributy, konstruktory a metody, popřípadě jejich parametry. Tyto dílčí části byly začleněny do balíčku *javaspec.parts*.

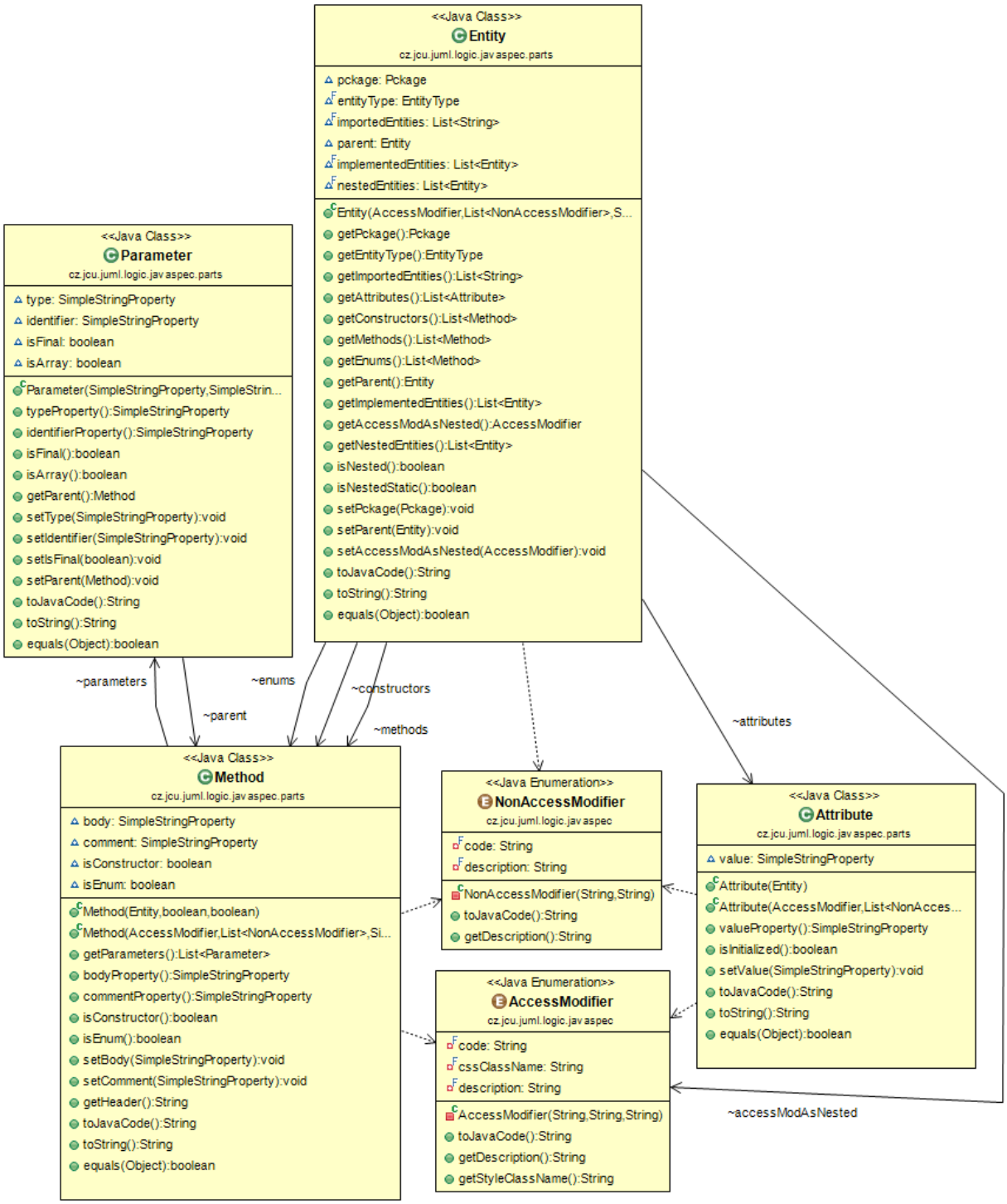
Dalšími vlastnostmi programovacího jazyka se stávají modifikátory přístupu a nepřístupu[23] a datové typy. Ty zde vystupují jako výčtové třídy, které jednotlivé třídy využívají.

Základní jádro aplikace se tedy skládá z těchto tříd:

- Entity,
- AccessModifier,
- NonAccessModifier,
- Attribute,
- Method,
- Parameter.

Na tuto základní strukturu se později nabalovaly další části, které lépe rozšířily aplikaci o požadované funkce.

[23] Tzv. non-access modifiers nebo také class variables přikládají atributům, metodám a třídám speciální vlastnosti [38] [39].



Obr. 5-1 Základní třídy jádra aplikace

5.2.2 Třídy a restrikce

K zajištění a znázornění vnitřní struktury jazyka Java byly do aplikace zahrnuty i základní typy tříd:

- Class,
- Abstract Class,
- Enumeration,
- Interface.

Sémantika jazyka Java však říká, že určité typy tříd nespecifikují některé části. Na základě tohoto zjištění musely být vytvořeny restrikce.

Třída *Restrictions* se tedy stará o zajištění těchto požadavků, aby nedocházelo ke zbytečným chybám na straně uživatele a tato omezení se tak dostala do jeho podvědomí. Poté co byla vytvořena tato třída, musely být rozeznávány i různé typy tříd. O to se stará výčtová třída *EntityType*, která posléze rozlišuje, jaká omezení jsou povolena za pomocí třídy *Restrictions.*

5.2.3 Instanční proměnné a metody

V reprezentaci instančních proměnných a metod je třeba uchovat jejich deklarace. Tudíž se skládají z modifikátoru přístupu, modifikátoru nepřístupu, datového či návratového typu a vlastního názvu. Instanční proměnné může být nastavena hodnota díky metodě *setValue().*

Metody v logické části aplikací disponují rozšířenými vlastnostmi. Ve třídě *Method* rozeznáváme, jestli se jedná o konstruktor či výčet metodami *isConstuctor()* a *isEnum()*, které se využívají při generování hlaviček kódu a při grafické reprezentaci modelované entity. Tou nejdůležitější částí této třídy se stává metoda *setBody(String)*, která dovoluje ‚upravovat' tělíčka vytvořených metod. Tuto metodu využívá zejména editor zdrojového kódu v GUI aplikace.

Třída *Parameter* poté představuje pole vytvářených parametrů dané metody. Jelikož tatáž metoda může mít různý počet parametrů, byla tomu přizpůsobena i tato třída. V instanční proměnné se tedy uchovává reference na vytvořenou metodu či výčet, která jasně definuje, ke kterému elementu parametr patří.

```
public class Parameter extends Part implements Serializable {

    private static final long serialVersionUID = 1L;

    transient SimpleStringProperty type;
    transient SimpleStringProperty identifier;
    boolean isFinal;
    boolean isArray;
    Method parent;

    public Parameter(SimpleStringProperty type, SimpleStringPr
        this.type = type;
        this.identifier = identifier;
        this.isFinal = isFinal;
        this.parent = parent;
    }
```

5.2.4 Rozšíření jádra

Základní struktura logického jádra zajistila funkcionalitu mezi jednotlivými třídami. Některé aspekty aplikace však vyžadovaly širší přístup a tudíž muselo být jádro rozšířeno o nespočet dalších tříd.

Nejprve bylo nutné začlenit třídy s mateřskými atributy do třídy *Member*, která se stará o přidělení požadovaného datového typu třídám *Attribute* a *Method* a stará se také o začlenění rodičovské entity. Celek poté uzavírá abstraktní třída Element, která reprezentuje každý element v jazyce Java, jimiž jsou Entity a jejich členy se svými vlastními názvy.

Generování programového kódu

Simulování struktury jazyka Java bylo již založeno a dalším milníkem bylo zajištění převodu jednotlivých elementů do programovacího kódu. Z návrhu vyplynulo, že bude vytvořeno rozhraní *Codeable*, které bude elementům a jejich členům poskytovat metodu *toJavaCode()*.

Aby bylo efektivní jednotlivé elementy převádět do kódu, byla vytvořena pomocná třída *LogicUtils*, která disponuje širokou škálou pomocných metod právě na převod do zdrojového kódu. Jedná se o metody, které například parsují prvky z kontejneru List do zdrojového kódu, či na základě libovolného počtu argumentů v parametru metody[24] sestavují textový řetězec.

[24] V programovací terminologii se jedná o vlastnost zvanou varargs, která dovoluje metodám vlastnit libovolný počet parametrů. Tyto parametry jsou posléze přístupné jako pole [40].

Spravování entit

Po naprogramování logické části aplikace nastala příprava na část grafickou. Podle hrubého náčrtu bylo jasné, že objekty, které budou umisťovány na pracovní plochu, musí být nějakým způsobem organizovány i v samotné logice aplikace.

K tomuto účelu slouží třída *EntityManager*, která se stará jak o samotné vytváření logických entit, tak i o jejich odebírání. Tato třída má však ještě další důležitou funkci, a tou je udržování jednotlivých entit v serializačním procesu.

5.3 Grafická část aplikace

Zkušenější vývojáři nejrůznějších aplikací mi dají jistě za pravdu, že nejpracnější částí vývoje se stává právě tvorba grafického uživatelského rozhraní. Na rozdíl od pokročilejších vývojových prostředí se GUI této aplikace snaží docílit co možná nejmenšího zatěžování uživatele grafickými prvky. I když se může zdát, že za nejdůležitější část aplikace se považuje právě její jádro, opak je pravdou a grafická část má nedílnou zásluhu na funkcionalitě samotné aplikace.

O tom pojednávají právě následující kapitoly, ve kterých se uvádí, jak byla vytvářena grafická část aplikace a jaké nástroje k tomu byly použity.

5.3.1 Modelování GUI

Každá webová či koncová aplikace by měla poskytovat intuitivní uživatelské prostředí. Problém však nastává v jeho vytvoření. Dříve se muselo spoléhat čistě na knihovny a repositáře programovacího jazyka a celý proces vývoje takového GUI byl časově náročný. Poté se za účelem rychlého modelování těchto prvků a následné převádění do zdrojového kódu začaly objevovat první editory.

V rozvoji jazyka Java se postupně objevovaly knihovny, které dopomáhaly vývojářům programovat vlastní GUI komponenty, ať už od prvních verzí třídy java.awt až po současný javax.swing. Nejnovějším přírůstkem, který přišel s verzí Java 8 a hraje velkou roli v této aplikaci, se stává knihovna JavaFX.

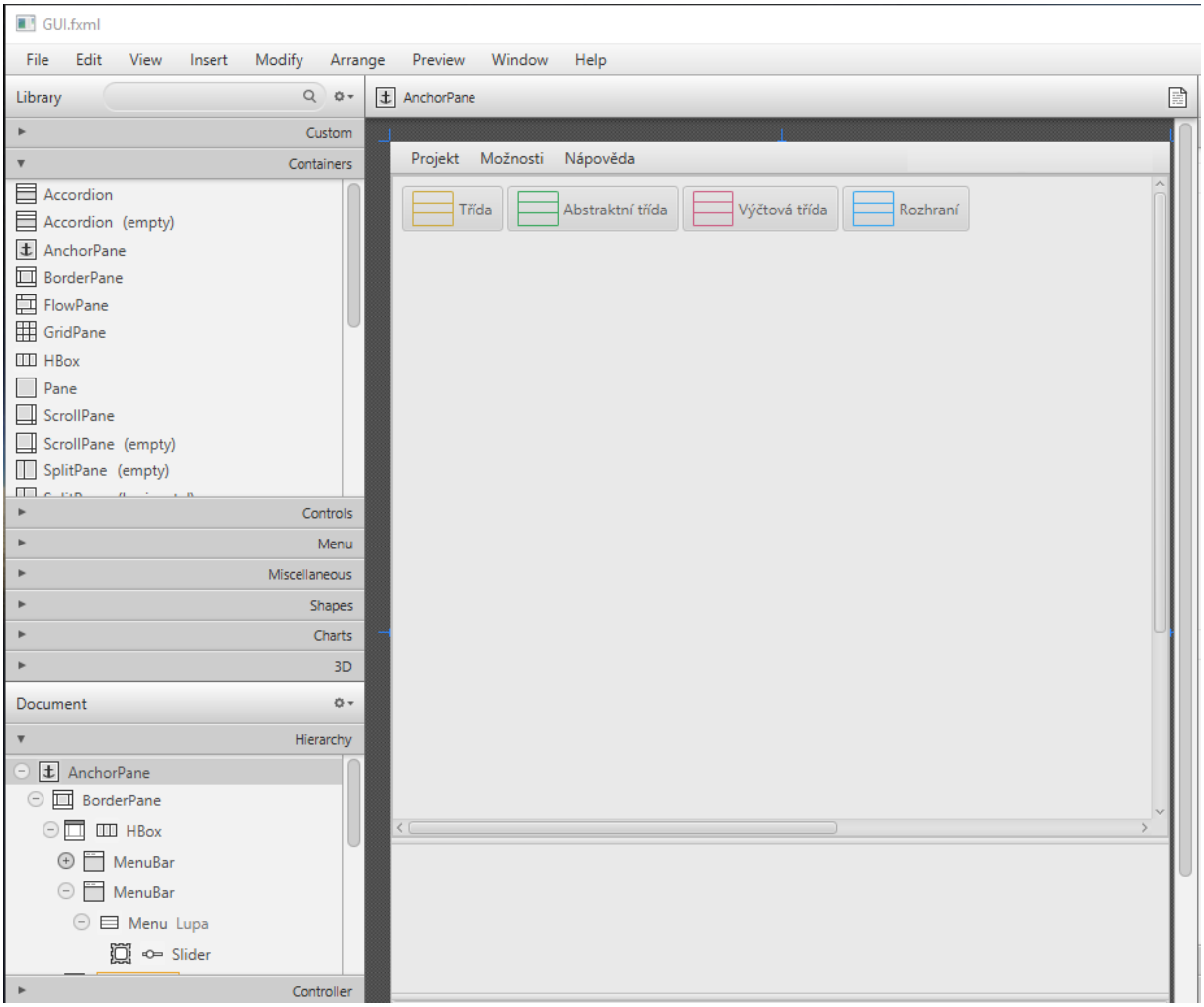
Za použití nástroje Scene Builder bylo možné pomocí technologie JavaFX vybudovat nové uživatelské prostředí.

Layout

Hlavním cílem v tomto prostředí byl celkový layout aplikace. Prvotní pokus o jednoduchý layout se nezdařil a testovací skupině přišel složitý, tudíž jsem musel vymyslet lepší variantu (viz Testování). Tuto verzi jsem nazval jako Lite, protože obsahovala pouze tyto části:

- Menu bar,
- Pracovní plocha,
- Editor kódu.

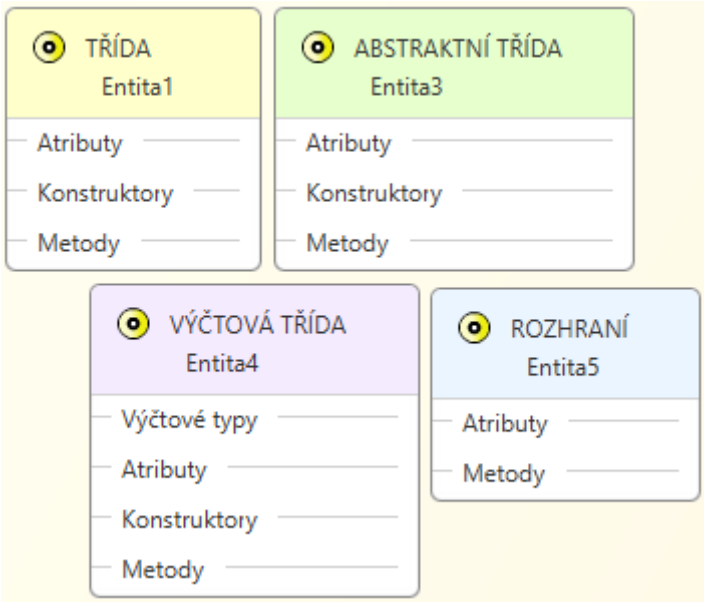
Panel nástrojů zde vystřídala jednoduchá tlačítka, která reprezentovala různé typy tříd. Pouhým přetáhnutím zvolené třídy na pracovní plochu se vytvořila grafická entita. Později bylo přidáno i terminálové okno.



Obr. 5-2 Grafické uživatelské rozhraní v nástroji Scene Builder

5.3.2 Grafické znázornění entity

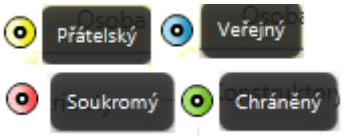
Základní ovládací komponenta této aplikace byla sestavována z dílčích částí, které reprezentují logickou strukturu jazyka Java. Na základě typu třídy disponovala oddíly, jako tomu je u diagramu tříd.



Obr. 5-3 Znázornění základních typů entit

Modifikátory přístupu

Již u samotného návrhu jsem uvažoval o nějaké vhodné grafické reprezentaci přístupových modifikátorů, až jsem vytvořil[25] jednoduchou ikonku ve tvaru oka, která představuje pohled na danou třídu. Po kliknutí na ikonku se změní její přístupnost.

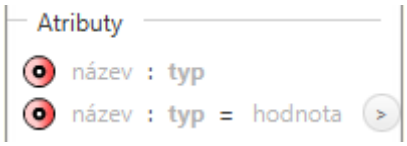


Obr. 5-4 Modifikátory přístupu

Atributy

Znázornění instanční proměnné v grafické formě bylo uskutečněno, jak vyplývá z návrhu, podle zápisu atributů z UML diagramu tříd.

[25] Za pomoci staré verze Macromedia Flash 8.



Obr. 5-5 Zápis atributů podle UML diagramu tříd

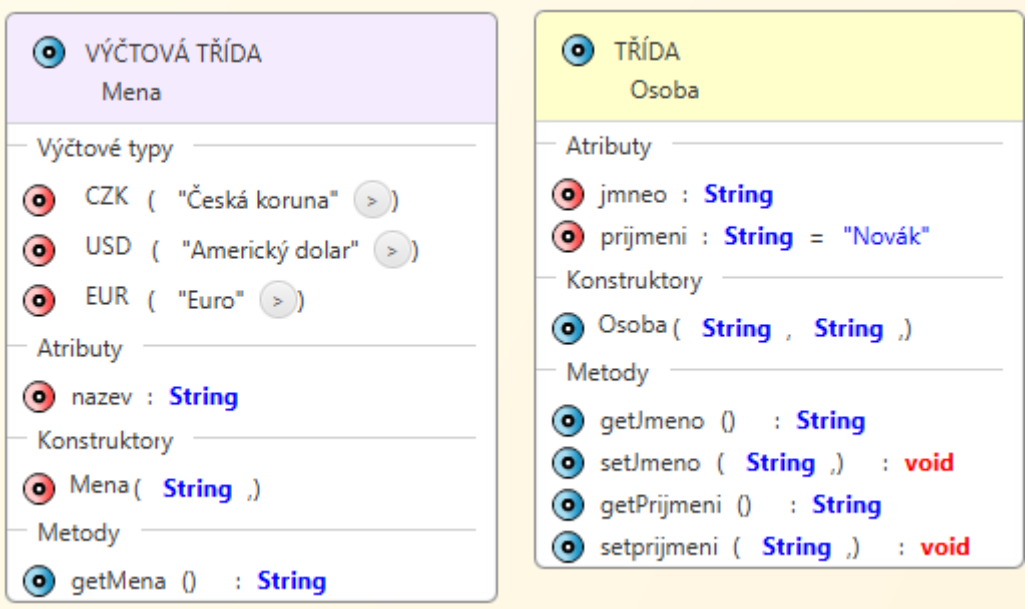
Metody, konstruktory a výčtové typy

Vytváření konstruktorů je velmi jednoduché, protože název konstruktoru se mění podle názvu třídy, tudíž dává uživateli najevo, že se jedná o incializátor dané třídy.

Metody na rozdíl od atributů jsou opatřeny návratovým datovým typem, který určuje, zda se jedná o getter či setter metodu.

Jednu věc však mají konstruktory a metody společnou, a to jsou parametry, které jim může uživatel přidělit. Graficky se poté znázorní datový typ parametru.

Speciálním případem jsou výčtové typy, které se uživateli zobrazí jako konstanty s možností předání vlastních parametrů. K tomu má možnost vytvořit instanční proměnné a soukromý konstruktor.



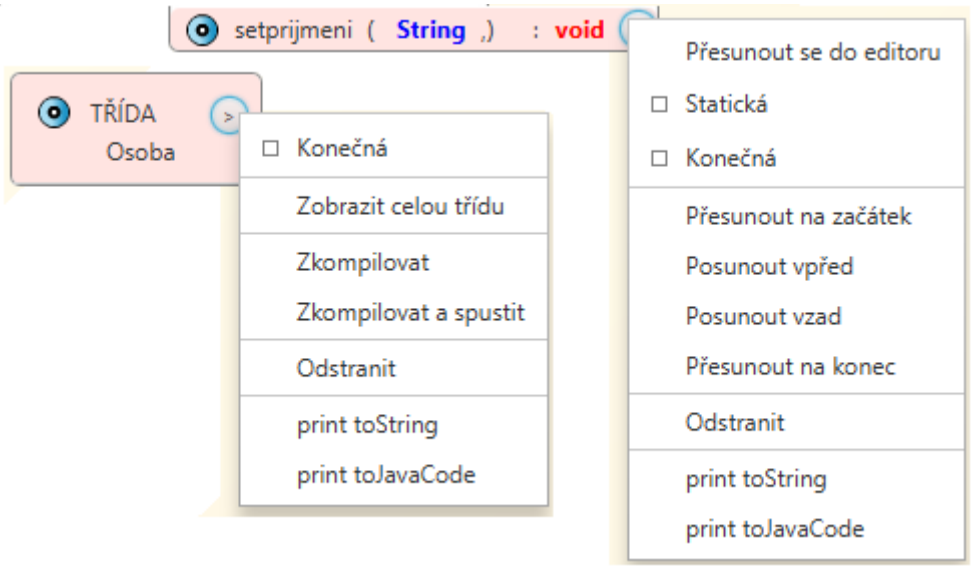
Obr. 5-6 Porovnání výčtové a obyčejné třídy

Rozšířené vlastnosti entity

Jak samotná entita, tak i její části disponují rozšířenými vlastnostmi, ať už se jedná o samotné přidávání jednotlivých členů nebo jejich mazání, přesouvání a jiné funkce. Proto bylo těmto elementům zřízeno tlačítko s kontextovým menu, které rozšiřuje jejich funkcionalitu.

Každý takto vytvořený element nese svoji kontextovou nabídku, která se liší podle typu elementu. Například entita má přidanou možnost se zkompilovat či zobrazit svůj obsah ve zdrojovém kódu jazyka Java.

Instanční proměnné se mohou v entitě přesouvat nahoru či dolů a můžeme měnit i pořadí parametrů metod a konstruktorů. Samotné konstruktory a metody poté zdědily vlastnost editace svých tělíček.



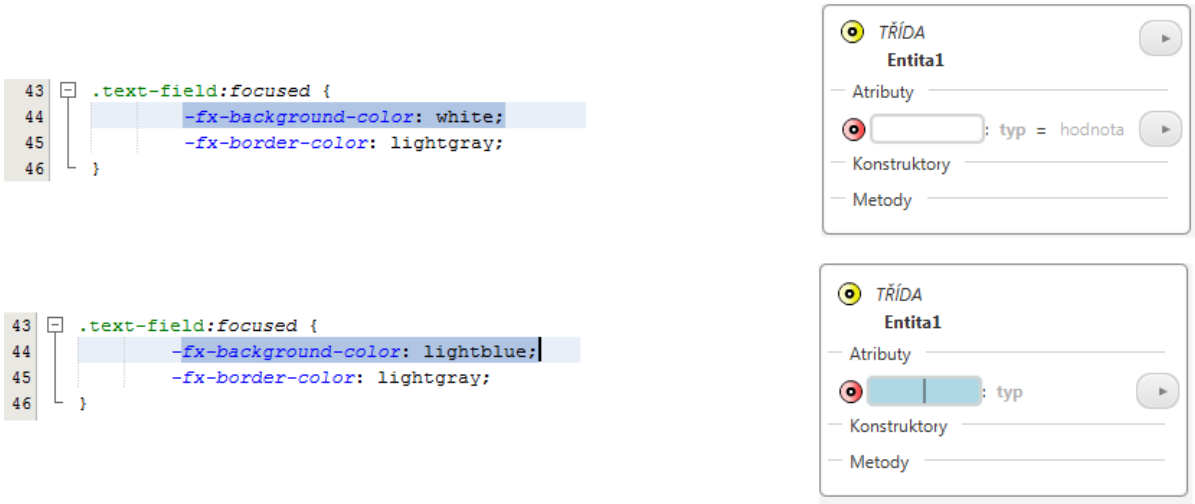
Obr. 5-7 Možnosti kontextové nabídky entity a metody

5.3.3 Efekty a CSS

Zřejmě tou největší novinkou, kterou technologie JavaFX disponuje, je možnost přidávání různých efektů, animací a jiných multimediálních přídavků. Některé prvky v této aplikaci jsou zvýrazňovány právě pomocí kaskádových stylů a metod knihoven JavaFX.

Jako první se musela zvýrazňovat syntaxe v entitě. Toho bylo docíleno pomocí vytvořených kaskádových stylů a metody *setStyle(String value)*. Hodnotu této metody představuje barevné

označení, které bylo dáno výčtovým typům ze třídy DataType. Samotné podbarvení textových polí bylo uskutečněno pomocí CSS.



Obr. 5-8 Využití kaskádových stylů v aplikaci

Zjednodušení entity

Mým záměrem též bylo, aby samotná entita vypadala čistě. Problém nastal, když se nahromadila kontextová tlačítka. Musel jsem tedy začít rozšiřovat tzv. listeners[26]. Jednalo se hlavně o posluchače, kteří museli reagovat na přejetí myši nad elementem a tím zobrazili kontextové tlačítko.

Další formou interakce entity jsou přítomné popisky kontextových tlačítek či modifikátorů přístupu. Opět se pouhým přejetím myši nad součástkou zobrazí její popisek (Obr. 5-4).

5.3.4 Správa grafických komponent

Třída zodpovědná za vytváření a správu grafických entit nazvaná GUIManager se též stará o propojení grafické a logické části. Nedílnou součástí této třídy je také zaregistrování a nastavení hlavní scény a především kontrolorů.

[26] Jedná se o posluchače událostí, které mají v jazyce Java vlastní specifikaci. Některé z nich pracují v reálném čase a aktualizují akce provedené uživatelem [41].

Kontroloři

Hlavní roli ve spuštění aplikace hrají tzv. controllers, kteří mají na starost načtení FXML souborů a správu oken. V této aplikaci je kontrolorů několik, mezi nimi jsou i GUIController a GUIEditorController. Právě poslední zmiňovaný se stará o vytvoření editoru kódu.

5.4 Implementace editoru kódu

Nejzajímavější funkcí této aplikace je rozhodně editor kódu, který dovoluje upravovat konstruktory a metody. Ve starších verzích Javy, kdy se ovládací prvky skládaly pomocí knihoven Swing, bylo možné naimportovat jednoduché API pro zvýraznění syntaxe. U technologie JavaFX mi s tímto pomohla knihovna RichTextFX.

Ve zkratce se pomocí této knihovny nadefinuje regulární výraz pro syntaxi jazyka Java. Poté se nechají jednotlivé části zvýraznit pomocí kaskádových stylů. K tomuto účelu slouží třída JavaCodeEditor, která tyto vlastnosti přejímá a modifikuje.

```
private static final Pattern PATTERN = Pattern.compile(
        "(?<KEYWORD>" + KEYWORD_PATTERN + ")"
                + "|(?<PRIMITIVE>" + PRIMITIVE_PATTERN + ")"
                + "|(?<PAREN>" + PAREN_PATTERN + ")"
                + "|(?<BRACE>" + BRACE_PATTERN + ")"
                + "|(?<BRACKET>" + BRACKET_PATTERN + ")"
                + "|(?<SEMICOLON>" + SEMICOLON_PATTERN + ")"
                + "|(?<STRING>" + STRING_PATTERN + ")"
                + "|(?<COMMENT>" + COMMENT_PATTERN + ")"
                + "|(?<GENERICS>" + GENERICS_PATTERN + ")"
                + "|(?<REFERENCE>" + REFERENCE_PATTERN + ")"
        );
public static StyleSpans<Collection<String>> computeHighlighting(String text) {
    Matcher matcher = PATTERN.matcher(text);
    int lastKwEnd = 0;
    StyleSpansBuilder<Collection<String>> spansBuilder
    = new StyleSpansBuilder<>();
    while(matcher.find()) {
        String styleClass = matcher.group("KEYWORD") != null ? "keyword" :
                            matcher.group("PRIMITIVE") != null ? "primitive" :
                            matcher.group("PAREN") != null ? "paren" :
                            matcher.group("BRACE") != null ? "brace" :
                            matcher.group("BRACKET") != null ? "bracket" :
                            matcher.group("SEMICOLON") != null ? "semicolon" :
                            matcher.group("STRING") != null ? "string" :
                            matcher.group("COMMENT") != null ? "comment" :
                            matcher.group("GENERICS") != null ? "generics" :
                            matcher.group("REFERENCE") != null ? "reference" : "primitive";

        spansBuilder.add(Collections.emptyList(), matcher.start() - lastKwEnd);
        spansBuilder.add(Collections.singleton(styleClass), matcher.end() - matcher.start());
        lastKwEnd = matcher.end();
    }
    spansBuilder.add(Collections.emptyList(), text.length() - lastKwEnd);
    return spansBuilder.create();
}
```

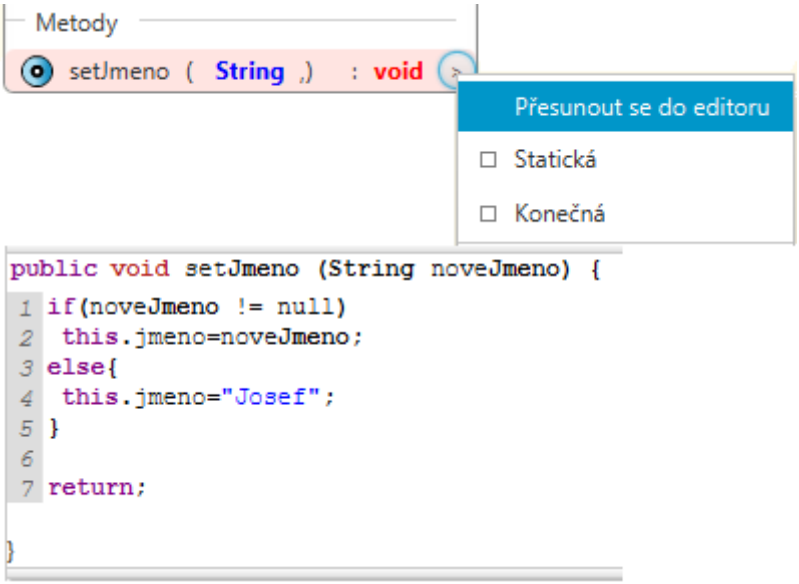
Obr. 5-9 Část kódu ze třídy JavaCodeEditor[27]

[27] Zdrojový kód byl použit ze zdrojových příkladů knihovny RichTextFX [28] a dále modifikován.

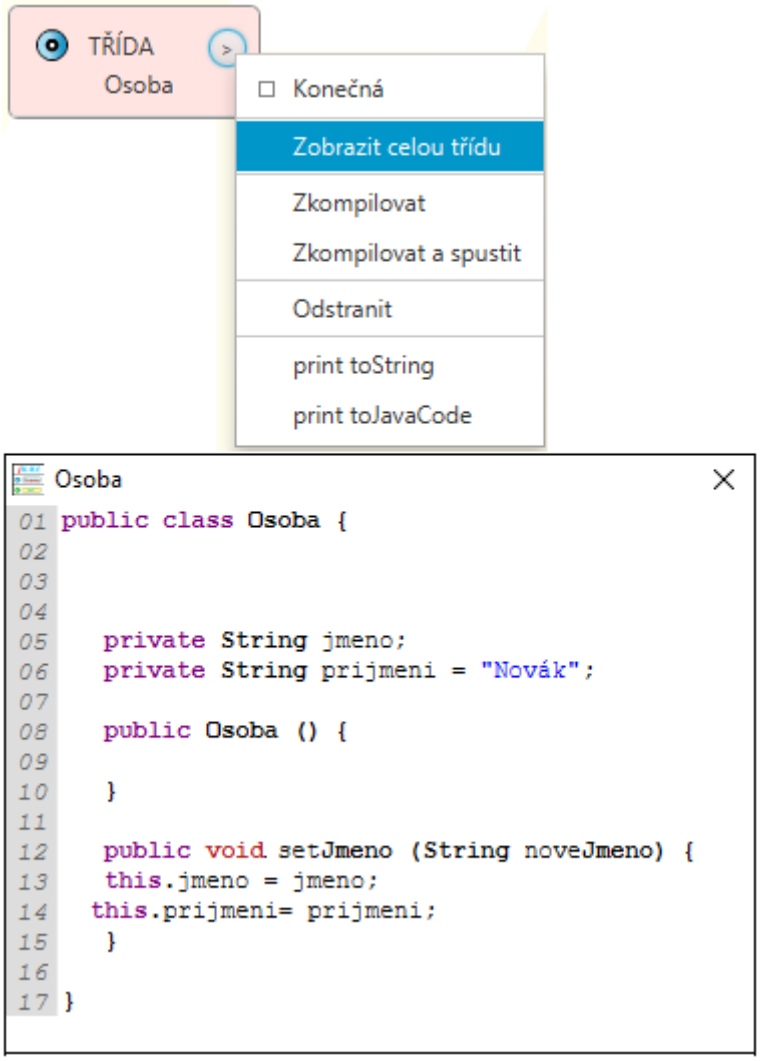
5.4.1 Možnosti editoru

Uživatel si většinu částí modelované třídy vytváří sám pomocí samotné entity a nemusí se tak starat o jejich programování, ale mít možnost editovat některé namodelované části a dopsat kód je velice žádané.

Proto bylo uživateli umožněno editovat hlavně tělíčka metod a konstruktorů, a to právě pomocí třídy JavaCodeEditor. Hlavička, kterou si uživatel nadefinoval modelováním v entitě, zůstává needitovatelná, aby nedošlo k porušení struktury projektu. V rozšířené nabídce metody či konstruktoru se poté uživatel přesune do editoru, který ukládá napsaný kód v reálném čase.



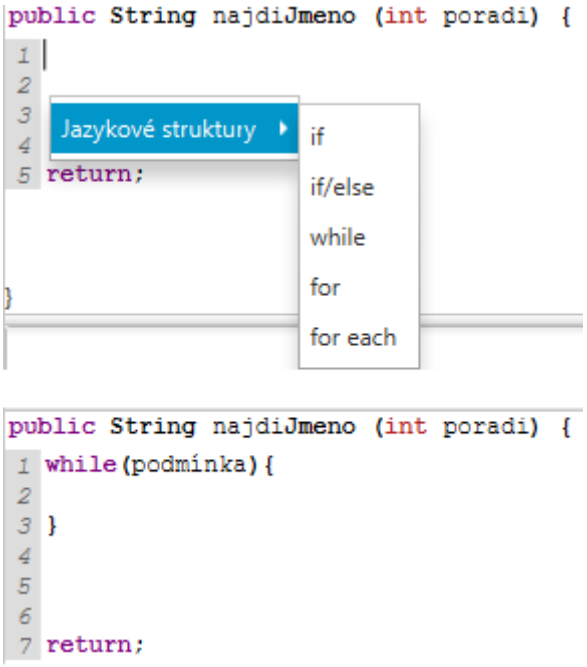
Další funkcí, kterou plní editor kódu, je možnost zobrazení celé třídy v možnostech entity. Uživateli se zobrazí samostatné okno se zdrojovým kódem celé třídy, aby ji mohl prohlížet a zkoumat samotnou syntaxi jazyka Java.



Obr. 5-10 Zobrazení celé třídy v editoru

Jazykové struktury

V zadání bakalářské práce stojí, že by měla aplikace obsahovat základní jazykové struktury. Toho jsem docílil právě v editoru kódu, kde pomocí kliknutí pravého tlačítka vyjede opět kontextové menu s výběrem základních řídících struktur. Tyto struktury jsou již předdefinovány a ušetří tak uživateli čas s jejich zápisem. Do budoucna bude aplikace rozšířena o další jazykové struktury.



Obr. 5-11 Přidávání řídících struktur

5.5 Implementace kompilátoru

Nejzásadnější funkcí této práce bylo převést vymodelované entity a jejich atributy i s uživatelským zdrojovým kódem do reprezentace jazyka Java. K tomuto účelu byl využit nástroj, kterým disponuje samotná Java již od verze 6 - Java Compiler [27].

Třídy, které jsou zodpovědné za kompilaci jednotlivých entit, se v tomto projektu nazývají *Compilerr* a *IOUtils*. Pomocná třída *IOUtils* disponuje metodami pro správu vstupních a výstupních souborů, které při kompilaci vznikají, přičemž samotná třída *Compilerr* vyvolává JavaCompiler pro kompilaci entit do souborů class.

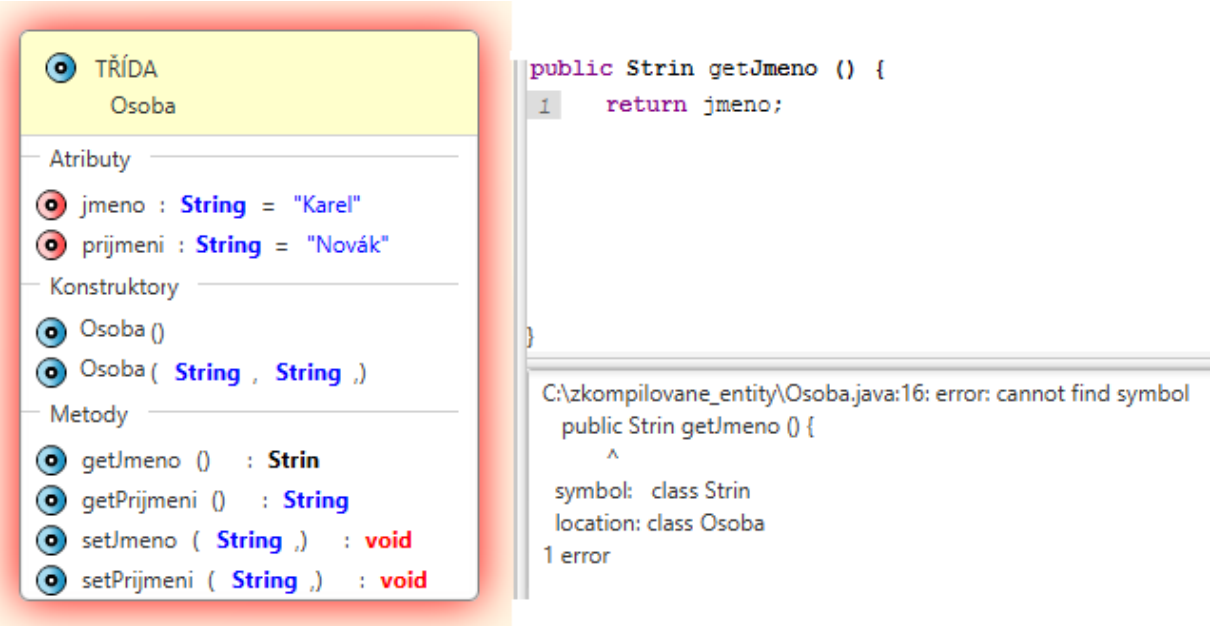
5.5.1 Možnosti kompilátoru

Jelikož byl do logiky aplikace implementován kompilátor, je možné aplikaci rozšiřovat o další funkce. Editor kódu, který doposud pracoval jako pasivní nástroj[28], byl obohacen o širší využití.

Při kompilaci entity je zahrnut i zdrojový kód, který uživatel vytvořil při psaní těla metody. V případě, že nastane při kompilaci problém, bude vytvořen výstupní soubor s chybami. Tento

[28] Uživatel mohl zapisovat vlastní kód, avšak bez zpětně vazby.

přístup je však uživateli k ničemu, proto bylo vytvořeno okno událostí[29]. Díky tomu má uživatel možnost se rychle přesvědčit, kde nastala chyba a ihned ji opravit.



Obr. 5-12 Zachycení chyby při editaci entity

Při kompilaci entity byl přidán i grafický efekt, znázorňující úspěšnost kompilace. Pokud se v dané entitě nachází chyba, zvýrazní se červeně, jestliže kompilace proběhla v pořádku, zvýrazní se zeleně.

Tato implementace si však od uživatele žádá přístup k vývojovému balíčku jazyka Java[30]. Tento předpoklad je vyžadován po uživateli při spuštění aplikace, aby bylo možné zajistit kompilaci entit.

5.6 Ukládání a načítání projektu

Nedílnou součástí každého editoru je možnost uložit a načíst svou práci. Taktéž je tomu i v této aplikaci. Jak již bylo zmíněno v návrhu (*viz kapitola Ukládání a otevírání projektu*) jednotlivé složky se musely převést do serializačního procesu. Za tuto práci zodpovídá třída *Serializer*, která všechny prvky na pracovní ploše ukládá, či načítá do souboru.

[29] V informatice se častěji používá výraz log, pro záznam výstupních dat.

[30] JDK ve verzi 1.8.0 a vyšší

K tomu, aby bylo možné všechny prvky uložit, musela být rozšířena logika aplikace o serializační klauzuli, kterou implementují ty logické části aplikace, které nepodléhají obecné serializaci[31]. Za tímto účelem vznikla třída *Part*, která přenáší tuto klauzuli ostatním členům.

[31] To se zejména projevuje u propojení grafické a logické části aplikace, kde se některé objektové části musejí rozložit do jednodušší formy, aby je bylo možné uložit.

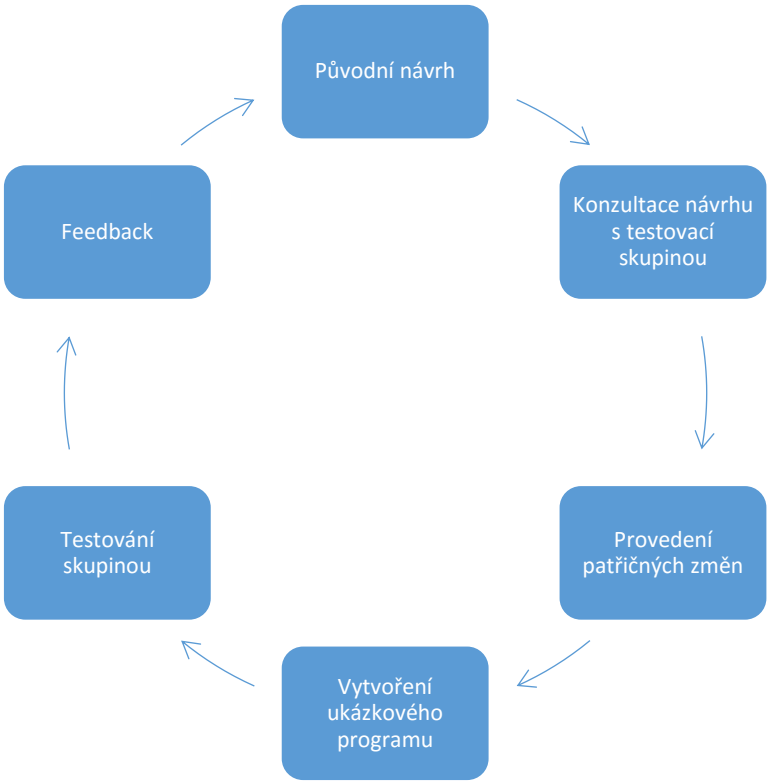
6 Testování aplikace

Při vývoji aplikace vzniká mnoho nápadů, ale i problémů, které ani sám autor práce není schopen zpozorovat. Vhodným způsobem, jak dojít k výsledné vizi, je zpětně analyzovat již navržené koncepty a konzultovat je s cílovou skupinou.

Mým záměrem bylo vytvořit užitečný nástroj, který by pomohl mladým lidem s programovacím jazykem Java. Za doby svého studia jsem byl vyhledáván a žádán svými kolegy o případné doučování tohoto jazyka. Díky této praxi se utvořila skupinka mladých lidí, která byla ochotna sdílet své názory a nápady při vývoji této aplikace.

6.1 Proces testování

Testovací fáze měla několik kroků, které mi pomohly při objasnění mých cílů a dosažení jednotlivých milníků[32].



Testovací skupina byla přítomna při prvotním návrhu a poskytla cenné rady a tipy, co v aplikaci zlepšit.

[32] Podobně jako znázorňuje Addie model.

Pravdivost svého tvrzení, které bylo vyřčeno na začátku této kapitoly, jsem si mohl rychle ověřit. Pří předložení původního návrhu testovací skupině jsem zjistil, že některé grafiké prvky by překážely a odrazovaly při skutečném modelování projektu. Například panel nástrojů toho neobsahoval mnoho a zbytečně zabíral místo a postrádal tak smysl. Po ujasnění priorit, co by měla aplikace obsahovat, jsem posléze skupině zasílal testovací verze.

7 Výsledky práce

Při vývoji vznikla velice rozsáhlá aplikace[33] s pracovním názvem jUML Designing Tool, která přináší nové možnosti návrhu projektů. V aplikaci byly rovněž vytvořeny projekty, které je zde možné otevřít a vyzkoušet si tak možnosti tohoto nástroje. Připraveny jsou i různé příklady, které nabádají uživatele k výběru.

7.1 Splnění cílů práce

Aplikace ve své podstatě odpovídá stanoveným cílům práce a pyšní se těmito přínosnými prvky:

- Grafické rozhraní dbá na přívětivost zacházení s ovládacími prvky a nerozptyluje uživatele zbytečnými nástroji.
- Pracovní plocha dovoluje spravovat nespočet vytvořených entit díky rolovací liště. Pro náročnější projekty byla zřízena funkce Zoom ke zlepšení čitelnosti na projektoru.
- Třída reprezentována jako box (entita), ve které se nacházejí jednotlivé prvky (instanční proměnné, konstruktory, metody, aj.). Práce spočívá v interakci mezi vytvořenými entitami a uživatelem.
- Aplikace umožňuje uživateli nejenom modelovat vlastní třídy, ale i upravovat zdrojový kód metod.
- Editace kódu probíhá v reálném čase, tudíž se neotevírají nová okna.
- Aplikace minimalizuje počet rizik vzniklých při zápisu obecné deklarace tříd a metod tím, že tyto aspekty sama generuje ve správném tvaru.
- Užitečný nástroj pro zpětnou vazbu uživateli poskytuje navržená konzole, která zobrazuje chyby vzniklé při editaci kódu.
- Jednotlivé části entity jsou graficky znázorněny podle ustálené sémantiky jazyka Java.
- Aplikace má schopnost ukládání a otevírání projektů.

7.2 Plánovaná vylepšení

Při vývoji aplikace byl kladen důraz na její možné rozšíření a vylepšení. Jako každý vývoj ani tento není ukončen, ale stále je doplňován aktualizacemi a opravami. Ze všech svých sil

[33] Aplikace čítá kolem 49 tříd, 10 balíčků, 6 externích knihoven. Před refaktorizací obsahovala aplikace přes 5 000 řádku kódu, a po ní přesně 3 524.

a zkušeností jsem se snažil pečovat o kvalitu a strukturu zdrojového kódu, který se tak stává snadno modifikovatelným.

Funkce, které bych rád v budoucnu zahrnul do této aplikace a které nebyly doposud zrealizovány:

- grafická implementace vztahů mezi jednotlivými entitami;
- pokročilejší zaměření na objektově orientovaný přístup;
- implementace balíčků a kontejnerů;
- multiselekce entit.

7.2.1 Připravené funkce

Některá rozšíření byla do aplikace již zahrnuta, nicméně nebyla řádně otestována. Ta se mohou v odevzdané verzi nakonec objevit:

- připravenost aplikace pro objektově orientovaný přístup,
- implementování spouštěcí funkce main a export do JAR souboru,
- možnost vkládání komentářů do zdrojového kódu a jiných struktur,
- exportování zdrojového kódu do formátu java.[34]
- generování Jar souboru.

7.3 Známé chyby

Žádná aplikace není dokonalá a i tato má bugy[35]. V současné době se mi do podvědomí dostaly následující chyby:

- Okno editoru se neroztahuje správně a občas není vidět hlavička metody.
- Kompilátor není schopný zkompilovat entitu typu Enum, kterou je nutno ručně doprogramovat prostřednictvím editoru kódu.

[34] Tato funkce umožňuje exportovaný formát využít i v jiném vývojovém prostředí.

[35] Slovem bug označujeme programátorskou chybu, která vzniká při vývoji aplikace. Do češtiny pak byl výraz přeložen jako moucha, či brouk [29].

7.4 Dostupnost a oprávnění

Aplikace vyvíjená v jazyku Java nese výhodu snadné dostupnosti a přístupnosti pro koncového uživatele bez nutnosti instalace[36]. Tato aplikace je vyvíjena jakožto svobodný software a spadá tedy pod licenci GNU General Public License (GPL).[37]

[36] K dostání je jako samostatně spustitelná aplikace, bez nutnosti instalace.

[37] Případné otázky směřujte na e-mail: chowik110@gmail.com.

8 Závěr

V rámci této bakalářské práce vznikla aplikace, která díky svým dvěma režimům návrháře a programátora vhodným způsobem doprovází začínající uživatele ve výuce programovacího jazyka Java.

Při vývoji vzniklo několik neobvyklých konceptů, které představují intuitivní nástroj pro vlastní návrh a vývoj projektů. Samotný uživatel uvítá svobodu při modelování vlastních objektů bez velkého seznamování se s prostředím aplikace. Inspirace vzniklá za pomocí UML diagramů umožňuje snadno modifikovat vytvářené entity a experimentovat s nimi.

Druhá složka aplikace pak vede uživatele ke správnému programování v jazyce Java. Editor kódu byl sofistikovaně navržen tak, aby dohlížel na správné dodržování syntaxe tohoto programovacího jazyka. Deklarační část tříd a metod se uživateli generuje automaticky, nicméně přítomnost chyb ze strany uživatele je z pedagogického hlediska žádaná, proto aplikace obsahuje kompilátor a okno chyb, které posléze upozorňují na vzniklé problémy. Uživatel má poté možnost upravovat zdrojový kód tělíček metod právě v editoru kódu a může využít i vytvořené šablony jazykových struktur.

Při programování aplikace jsem se setkal s komplikací týkající se vytváření vlastních grafických komponent, jelikož základní komponenty knihoven Swing i JavaFX nebyly vyhovující, musel jsem realizovat vlastní návrh. I přes náročnost tohoto úkolu jsem se naučil chápat nové inovativní postupy při vytváření grafických objektů a lépe porozuměl technologii JavaFX.

Tato aplikace ve své podstatě využívá nové technologie jazyka Java, tudíž počítám s možností rozšíření o další prvky, které byly zmíněny výše, popřípadě kolaboraci s dalšími vývojáři, kteří projeví o tuto aplikaci zájem.

9 Citovaná literatura

[1] ADDIE Model. *Training industry* [online]. © 2015 [cit. 2015-09-25]. Dostupné z: http://www.trainingindustry.com/wiki/entries/addie-model.aspx

[2] WYSIWYG. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, © 2001-2015 [cit. 2015-09-28]. Dostupné z: https://cs.wikipedia.org/wiki/WYSIWYG

[3] ROUSE, Margaret. WYSIWYG (what you see is what you get). In: *WhatIs.com* [online]. © 2011 [cit. 2015-10-02]. Dostupné z: http://whatis.techtarget.com/definition/WYSIWYG-what-you-see-is-what-you-get

[4] Vývojové prostředí. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, © 2001-2015 [cit. 2015-10-03]. Dostupné z: https://cs.wikipedia.org/wiki/Vývojové_prostředí

[5] CHRISTIANO, Marie. What are Integrated Development Environments?. *All About Circuits* [online]. © 2015, 1 [cit. 2015-10-15]. Dostupné z: http://www.allaboutcircuits.com/technical-articles/what-are-integrated-development-environments/

[6] What exactly is the meaning of an API?. In: *Stack overflow* [online]. © 2015 [cit. 2015-10-16]. Dostupné z: http://stackoverflow.com/questions/7440379/what-exactly-is-the-meaning-of-an-api

[7] Introduction To OMG's Unified Modeling Language®. *OMG* [online]. U.S.A (Needham): Object Management Group, © 1997-2015 [cit. 2015-10-17]. Dostupné z: http://www.omg.org/gettingstarted/what_is_uml.htm

[8] *Class diagram: Atribut a jeho syntaxe* [online]. Liberec: Pavus, © 2005 [cit. 2015-10-20]. Dostupné z: http://mpavus.wz.cz/uml/uml-s-class-3-3-1.php

[9] MARTIN, Robert. *UML for Java programmers* [online]. 1. Upper Saddle River, NJ: Prentice Hall PTR, 2003, s. 22-28, xxiv, 249 p. [cit. 2015-10-20]. ISBN 01-314-2848-9.

Dostupné z: http://www.csd.uoc.gr/~hy252/references/UML_for_Java_Programmers-Book.pdf

[10] *OMG Unified Modeling Language*. 2.5. Needham (Massachusetts): Object Managment Group, © 2015. Dostupné také z: http://www.omg.org/spec/UML/2.5/

[11] *NetBeans IDE: The Smarter and Faster Way to Code* [online]. California: Oracle America, © 2015 [cit. 2015-10-10]. Dostupné z: https://netbeans.org/features/index.html

[12] *About BlueJ* [online]. Canterbury: Michael Kolling, © 2015 [cit. 2015-10-12]. Dostupné z: http://www.bluej.org/about.html

[13] About StarUML. *StarUML* [online]. © 2005 [cit. 2015-10-15]. Dostupné z: http://staruml.sourceforge.net/v1/about.php

[14] ČÁPKA, David. Layout. In: *IT network* [online]. Praha 3: IGNUM, © 2015 [cit. 2015-10-21]. Dostupné z: http://www.itnetwork.cz/html-css/html-manual/rozlozeni/html-layout-rozlozeni-stranky-cesky-manual/

[15] How to Use Menus. *Oracle* [online]. California: Oracle Corporation, © 2015 [cit. 2015-10-22]. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

[16] *The Java Language Specification*. 7th edition. U.S.A. (California): Oracle America, © 2011. Dostupné také z: https://docs.oracle.com/javase/specs/jls/se7/jls7.pdf

[17] Understanding Class Members. *Oracle* [online]. California: Oracle Corporation, © 2015 [cit. 2015-11-02]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

[18] ECKEL, Bruce. *Thinking in Java* [online]. 3rd ed. Upper Saddle River, N.J.: Prentice Hall, 2003, Kapitola 4 [cit. 2015-11-03]. ISBN 01-310-0287-2. Dostupné z: http://www.mindviewinc.com/Books/downloads.html

[19] KROLIK, Aaron. How do you explain getters and setters in Java to a beginner?. In: *Quora* [online]. California: Quora, © 2013 [cit. 2015-11-06]. Dostupné z: https://www.quora.com/How-do-you-explain-getters-and-setters-in-Java-to-a-beginner

[20] Creating and Using Packages. *Oracle* [online]. California: Oracle Corporation, © 2015 [cit. 2015-11-15]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/package/packages.html

[21] *Java™ Platform, API Specification: Interface Serializable*. 6th ed. California: Oracle Corporation, © 1993-2015. Dostupné také z: http://docs.oracle.com/javase/6/docs/api/java/io/Serializable.html

[22] TEICHMANN, Radek. Serializace a její problémy. In: *Vsaď na Javu* [online]. Brno: MoroSystems, © 2006-2015 [cit. 2015-11-19]. Dostupné z: http://vsadnajavu.cz/2009-09/odborne/java-j2ee/serializace-a-jeji-problemy/

[23] AIKEN, Alex. Compilers. *Coursera* [online]. California: Coursera, © 2015 [cit. 2015-11-20]. Dostupné z: https://class.coursera.org/compilers/lecture/preview. In association with Stanford University.

[24] JavaFX Overview. *Oracle* [online]. California: Oracle Corporation, © 2008-2014 [cit. 2015-12-05]. Dostupné z: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

[25] JavaFX Scene Builder: User Guide. *Oracle* [online]. California: Oracle Corporation, c2008-2014 [cit. 2015-12-05]. Dostupné z: http://docs.oracle.com/javase/8/scene-builder-2/user-guide/index.html

[26] ControlsFX. *FX Experience* [online]. 2015 [cit. 2015-12-08]. Dostupné z: http://fxexperience.com/controlsfx/

[27] *Java™ Platform, API Specification: Interface JavaCompiler*. 7th ed. California: Oracle Corporation, c1993-2016. Dostupné také z: https://docs.oracle.com/javase/7/docs/api/javax/tools/JavaCompiler.html

[28] MIKULA, Tomáš. RichTextFX. *GitHub* [online]. 2015 [cit. 2015-12-10]. Dostupné z: https://github.com/TomasMikula/RichTextFX

[29] Programátorská chyba. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2016 [cit. 2016-01-10]. Dostupné z: https://cs.wikipedia.org/wiki/Programátorská_chyba

[30] Význam slova sémantika. *Lidový slovník* [online]. b.r. [cit. 2015-12-15]. Dostupné z: http://lidovyslovnik.cz/index.php?dotaz=s%E9mantika

[31] HOWE, Dennis. Look and feel. In: *Dictionary: The Free On-line Dictionary of Computing* [online]. Chicago Manual Style (CMS), © 2001 [cit. 2015-12-16]. Dostupné z: http://www.dictionary.com/browse/look-and-feel

[32] Význam frontend. *IT slovník* [online]. © 2008-2015 [cit. 2015-12-18]. Dostupné z: http://it-slovnik.cz/pojem/frontend

[33] NISHADHA, . Class Diagram Relationships in UML with Examples. In: *The Creately Blog* [online]. Melbourne: Cinergix Pty, © 2008-2015 [cit. 2015-10-10]. Dostupné z: http://creately.com/blog/diagrams/class-diagram-relationships/

[34] *UML Specification*. 1.4. Needham (Massachusetts): Object Managment Group, 2001, s. 3-35. Dostupné také z: http://www.omg.org/spec/UML/1.4/

[35] LIM, Winnie. A Beginner's Guide to Wireframing. In: *EnvatoTuts* [online]. Melbourne: Envato Pty, 2015 [cit. 2015-11-07]. Dostupné z: http://webdesign.tutsplus.com/articles/a-beginners-guide-to-wireframing--webdesign-7399

[36] *Java SE Documentation: JAR File Overview*. 7th Edition. California: Oracle Corporation, 1993-2015. Dostupné také z: http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jarGuide.html

[37] MIKULA, Tomáš. Tomas Mikula repositories. *GitHub* [online]. California: GitHub, 2015 [cit. 2015-12-20]. Dostupné z: https://github.com/TomasMikula

[38] Class variables and constants. BARNES, David a Michael KÖLLING. *Objects first with Java: a practical introduction using BlueJ*. 5th ed. Boston: Pearson, 2012, s. 190-195. ISBN 0132492660.

[39] BABLAD, Pramod. What Are Access And Non-Access Modifiers In Java?. *Java Concept Of The Day: Java Tutorial Site For Beginners* [online]. GoDaddy, © 2015 [cit. 2015-12-18]. Dostupné z: http://javaconceptoftheday.com/access-and-non-access-modifiers-in-java/

[40] Methods with a Variable Number of Parameters. HORSTMANN, Cay a Garry CORNELL. *Core Java*. 8th ed. Upper Saddle River, NJ: Prentice Hall/Sun Microsystems Press, 2008, s. 214-215. ISBN 0132354799.

[41] SINTES, Tony. Events and listeners. *JavaWorld* [online]. Framingham: JavaWorld, © 1994-2016 [cit. 2016-01-06]. Dostupné z: http://www.javaworld.com/article/2077351/java-se/events-and-listeners.html