Didaktické pomůcky pro výuku programování s využitím technologie Java

Bc. Martin Sup

Diplomová práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin Sup**
Osobní číslo:	**A16214**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Učitelství informatiky pro střední školy**
Forma studia:	**prezenční**
Téma práce:	**Didaktické pomůcky pro výuku programování s využitím technologie Java**
Téma anglicky:	**Lecture Notes for the Java Programming Course**

Zásady pro vypracování:

1. **Seznamte se s technologií Java v aktuální verzi.**
2. **Proveďte rešerši v oblasti dostupných výukových materiálů pro jazyk Java.**
3. **Prostudujte zvolený Rámcový vzdělávací program pro střední školy, zejména oblast výuky programování.**
4. **Zpracujte podklady pro výuku v rámci vhodně zvolených tematických celků.**
5. **Sestavte sadu ukázkových příkladů.**
6. **Navrhněte sadu testovací úloh včetně vzorového řešení.**
7. **Vyhodnoťte roli vytvořených materiálů v současné výuce na středních školách.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **PAVLÍČKOVÁ, Jarmila a Luboš PAVLÍČEK. Úvod do Javy. Praha: Oeconomica, 2005. ISBN 80-245-0963-6.**
2. **Rámcový vzdělávací program pro obor vzdělání 18-20-M/01 Informační technologie. Národní ústav odborného vzdělávání, Praha, 2008. Dostupné z: http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf**
3. **URMA, Raoul-Gabriel, Mario FUSCO a Alan MYCROFT. Java 8 in action: lambdas, streams, and functional-style programming. Shelter Island: Manning, 2015. ISBN 978-1617291999.**
4. **SCHILDT, Herbert. Java. Tenth edition. New York: McGraw-Hill Education, 2017. ISBN 978-1259589331.**
5. **AZIZ, Adnan, Tsung-Hsien LEE a Amit PRAKASH. Elements of Programming Interviews in Java: The Insiders' Guide. 2nd edition. CreateSpace Independent Publishing Platform, 2015. ISBN 978-1517671273.**
6. **BLOCH, Joshua. Effective Java. 2nd ed. Upper Saddle River, NJ: Addison-Wesley, c2008. ISBN 978-0321356680.**

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **1. prosince 2017**

Termín odevzdání diplomové práce: **16. května 2018**

Ve Zlíně dne 11. prosince 2017

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Jméno, příjmení: Bc. Martin Sup

Název bakalářské/diplomové práce: Didaktické pomůcky pro výuku programování s využitím technologie Java

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 16.5.2018

.....
podpis diplomanta

ABSTRAKT

Cílem práce je vytvořit výukové materiály pro předmět uvádějící žáky do programování v jazyce Java. Teoretická část je zaměřena na popis vybraných vývojových prostředí a obhájení volby cílového programovacího jazyka. Dále se zabývá historickým vývojem platformy Java a závěr teoretické části se věnuje vybrané výukové literatuře. Praktická se zaměřuje na volbu tematických celků, popisu zvoleného vývojového prostředí BlueJ a v druhé části jsou popsána a vysvětlena jednotlivá témata.

Klíčová slova: java programování bluej

ABSTRACT

The aim of the thesis is to create teaching materials for the subject that introduces students to Java programming. The theoretical part is focused on the description of selected development environments and the justification of the chosen programming language. Furthermore it looks into the historical development of the Java platform and the end of the theoretical part is devoted to the selected teaching literature. The practical part focuses on the selection of appropriate topics, a description of the chosen development environment BlueJ and the second part describes and explains the selected topics.

Keywords: java programming bluej

Chtěl bych tímto poděkovat Ing. Bc. Pavlu Vařachovi, Ph.D. za vedení mé diplomové práce a především za jeho trpělivost při jejím vypracovávání. Zároveň chci poděkovat celé rodině, která mi stála oporou během mého studia.

Prohlašují, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Cílem mé práce je vytvořit sadu tematických výukových materiálů, využitelnou ve výuce programování. Výuka programování je v současnosti součástí mnoha studijních oborů na českých středních školách. Výběr vyučovaného programovacího jazyka hraje svou roli, neboť dává studentovi lepší šance na přímý přechod do produkční oblasti programování. V první části se tedy věnuje obhajobě volby Javy, jako zvoleného programovacího jazyka a vývojové platformy jako celku. Při výuce není snadné zvolit správné vývojové prostředí pro účely výuky. V teoretické části tedy práce zkoumá výhody a nevýhody vybraných vývojových prostředí. Dále se teoretická část věnuje popisu verzí a implementovaných funkcí do platformy Java, aby byl k dispozici aktuální přehled pro účely další výuky. Závěr teoretické části věnuje pozornost přehledu využitelné výukové literatury programování v jazyce Java.

V praktické části je cílem určit vhodné tematické okruhy jako důležité opěrné body k výuce. Dále se praktická část věnuje popisu prostředí zvolené aplikace BlueJ, jež byla zvolena jako nejvhodnější pro počáteční výuku. Druhá polovina praktické části se zaměřuje na vysvětlení jednotlivých pojmů zvolených tematických okruhů včetně ukázek zápisu zdrojového kódu.

I. TEORETICKÁ ČÁST

1 VÝCHOZÍ RÁMEC KURZU

1.1 Cílová skupina

Ačkoliv by se dle mého názoru nemělo čekat s výukou algoritmizace a jisté úrovně programování na střední školu, vyučování konkrétního, v praxi využitelného a přitom jednoduchého a srozumitelného programovacího jazyka, kterým je Java, již lze zařadit do odborného středního vzdělání. Výuka tedy bude zaměřena především na studenty středních škol studující informační technologie.

Základním dokumentem, ze kterého lze vycházet při vytyčování rámce a cílů při vyučování programování a algoritmizace na střední škole je Rámcový vzdělávací program (dále jen „RVP“), zde konkrétně Rámcový vzdělávací program pro obor vzdělání 18 – 20 – M/01 Informační technologie, vydaný Ministerstvem školství, mládeže a tělovýchovy dne 29. 5. 2008. Oblast, která nás ze zvoleného RVP zajímá především je Programování a vývoj aplikací, která definuje učivo a výsledky vzdělávání, kterých mají studenti daného oboru dosáhnout. Vybraná oblast definuje 5 oblastí učiva, ze kterých nás pro účely tohoto kurzu zajímají první 3 z nich (viz Tabulka 1 obsahující pouze vybrané učivo).

Výsledky vzdělávání	**Učivo**
Žák: - zná vlastnosti algoritmu; - zanalyzuje úlohu a algoritmizuje ji; - zapíše algoritmus vhodným způsobem;	**1 Algoritmizace** - význam, prvky algoritmu
- použije základní datové typy; - použije řídící struktury programu; - vytvoří jednoduché strukturované programy;	**2 Strukturované programování** - datové typy - řídicí struktury
- rozumí pojmům třída, objekt a zná jejich základní vlastnosti; - použije jednoduché objekty;	**3 Úvod do objektového programování** - třída, objekt, vlastnosti tříd

Tabulka 1 – Učivo a výsledky vzdělávání v oblasti dle RVP [1]

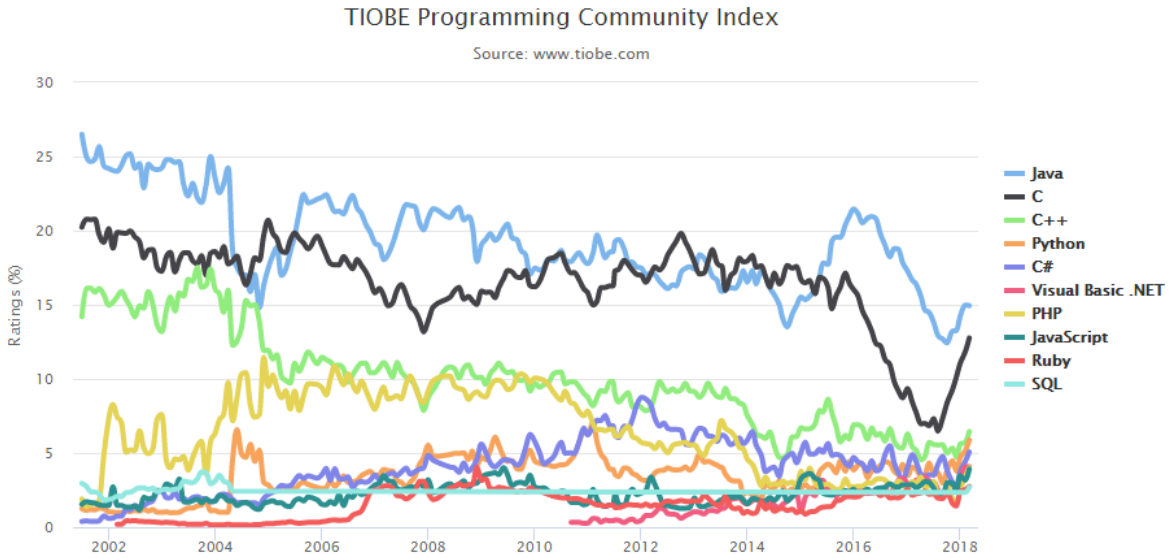
1.2 Volba vývojové platformy

Programovacích jazyků a technologií, které lze použít pro výuku programování je celá řada. Při volbě použitého programovacího jazyka, jako nástroje, vyplyne Java mezi předními kandidáty hned z několika důvodů. Jedná se především o velkou a aktivní komunitu, její celkovou rozšířenost, aplikovatelnost do reálného vývoje, poměrně nízké bariéry pro naučení a demonstraci důležitých konceptů jak strukturovaného, tak objektového programování nebo dostupnost literatury a internetových zdrojů.

1.2.1 Rozšířenost a popularita programovacích jazyků

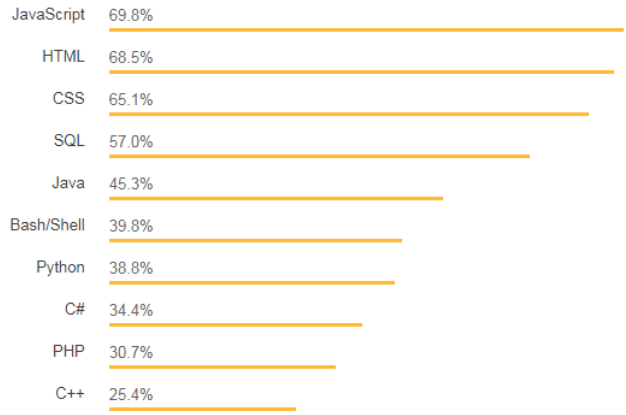
V současné době je mezi programovacími jazyky velká konkurence pro jejich velký počet. Platí ovšem to pravidlo, že mezi nejpopulárnější jazyky patří ty, které dovedou na požadované úrovni vytvořit produkt, který si trh žádá. Díky širokému uplatnění se i jazyk Java umisťuje pravidelně vysoko na žebříčcích oblíbenosti a výzkumech zjišťující mimo jiné rozšířenost různých programovacích jazyků.

V indexu popularity programovacích jazyků vydávaných společností TIOBE Software BV se v březnu 2018 umístila Java na prvním místě. Index vychází z četnosti vyhledávání programovacích jazyků v široké škále internetových vyhledávačů. [2]



Obrázek 1 – Vývoj popularity programovacích jazyků (2002-2018) [2]

Ve výzkumu prováděném portálem StackOverflow.com na vzorku 101 592 softwarových vývojářů ze 183 zemí umisťuje Javu v popularitě na 5. místo v kategorii „Programming, Scripting, and Markup Languages“, kde se výše umístila pouze čtveřice jazyků, které představují jedny z pilířů webových technologií. [3]



Obrázek 2 – Nejpoužívanější programovací jazyky respondentů Stack Overflow [3]

V neposlední řadě pak uvedu výzkum Git User's Survey z roku 2016, který vycházel z počtu 9 401 respondentů využívajících verzovací systém Git. Zde na otázku „Ve kterém programovacím jazyku jste zdatný?“ uvedlo 45 % respondentů, jež odpověděli, jako jeden z vybraných jazyk Java. [4]

1.2.2 Základní myšlenky Javy

Prvním, řekl bych nejstěžejnějším, principem technologie Java je přenositelnost. Tvůrci Javy si dali za cíl, aby program v tomto jazyce napsaný šlo spustit na jakékoliv počítačové platformě, na kterém běží tzv. běhové prostředí Javy, které slouží jako prostředník mezi spuštěnou aplikací a pod ní ležícím hardwarem a softwarem. Díky této tezi nemusí vývojář řešit mnohá úskalí spojená s vývojem pro dané zařízení, neboť tyto problémy již běhové prostředí řeší za něj. Taktéž počty a spektrum zařízení, pro která můžeme v současnosti na bázi technologie Java vyvíjet, je velmi široké. Dominantu tvoří především mobilní aplikace a tablety, dále pak serverové systémy, ale výjimkou nejsou ani desktopové aplikace.

V pořadí druhým heslem Javy představuje její jednoduchost, jež dovede studenta více vtáhnout do problematiky programování, neboť syntakticky vychází z rodiny programovacích jazyků založených na jazyku C, což mu garantuje velmi přehlednou a srozumitelnou sadu principů a gramatických pravidel.

Dalším rysem programování v Javě je její robustnost. Zápis zdrojového kódu v jazyce Java podléhá tzv. silnému typování (angl. strong typing), v čemž se liší od mnoha v současnosti taktéž populárních jazyků, jako jsou PHP nebo JavaScript, kde je datový typ proměnné automaticky odvozen, Java naproti tomu vyžaduje téměř vždy, aby byl datový typ každé

proměnné jasně deklarován předem, Toto umožní studentovi (resp. vývojáři) vždy vědět s jakým typem dat pracuje, jakou sadu operací lze nad těmito daty provádět a snáze pak zjistí, že se dopustil neúmyslné chyby, která by se běžně projevila až při běhu programu. [5]

A v neposlední řadě je Java významným zástupcem objektově orientovaných programovacích jazyků. Díky tomu lze vysvětlit pomocí Javy základní koncepty OOP i problémy s nimi spojenými a jak Java tyto problémy případně řeší nebo se jim vyhýbá.

1.2.3 Uplatnění technologie Java

Platforma Java patří mezi ty s velmi širokým záběrem. V současnosti nejpopulárnější uplatnění nachází v oblasti mobilních aplikací běžících na operačních systémech Android, kde ačkoliv Java není výlučnou variantou (mezi alternativy patří např. kombinace jazyků C a C++ nebo Go), dominuje mezi aplikacemi za použití Android SDK, tedy základní sady nástrojů pro vývoj. [6] To umožňuje vyvíjet aplikace pro 85% podíl, který Android drží na trhu chytrých telefonů. [7]

Javu taktéž využívají firmy celosvětově pro serverovou část své webové prezentace a to především prostřednictvím frameworků jako jsou například Spring MVC nebo JavaServer Faces (JSF), které značně usnadňují vývoj. [8]

Aplikace můžou být v jazyku Java naprogramovány i pro desktopové aplikace, u kterých se považuje za důležitou přenositelnost mezi různými zařízeními, využívaná jak ve firmách tak při vědecké činnosti.

V souvislosti s desktopovými aplikacemi lze taktéž použít Javu pro vývoj počítačových her. Nejznámějším představitelem takové hry je Minecraft. Obecně se ale hry v Javě vyvíjí pouze okrajově, protože v oblasti počítačových her je kladen důraz na výkon. V tomto aspektu mají jiné jazyky nad Javou stále nezanedbatelnou výhodu.

1.3 Vývojová prostředí

Abychom jako tvůrci programů a aplikací měli co nejméně práce, měli bychom využít služeb jednoho specializovaného druhu softwaru. Integrované vývojové prostředí (angl. Integrated Development Environment, zkráceně IDE) je počítačový software, který v sobě integruje řadu funkcí, kterých programátoři využívají, aby mohli své úsilí věnovat jiným problémům, než ty, od kterých je schopno toto prostředí oprostit. Mezi základní funkce každého IDE patří:

- **zvýrazněná syntaxe** – barevně odlišuje zásadní prvky zdrojového kódu, jakými jsou běžně klíčová slova (slova se speciálním významem), čísla, textové řetězce,
- **provádění kontroly správnosti syntaxe** – zvýrazňuje syntaktické chyby již během psaní zdrojového kódu, což umožňuje odhalit chyby mnohem dříve,
- **zajištění kompilace** – zprostředkovává překlad zdrojového kódu pomocí patřičného kompilátoru,
- **debugovací nástroje** – obsahuje nástroje, které jsou schopné pomoci programátorovi monitorovat aplikaci za běhu, zjišťovat stav aplikace a jejich prvků a odhalit tak chyby, které nastaly až poté, co aplikace začala běžet

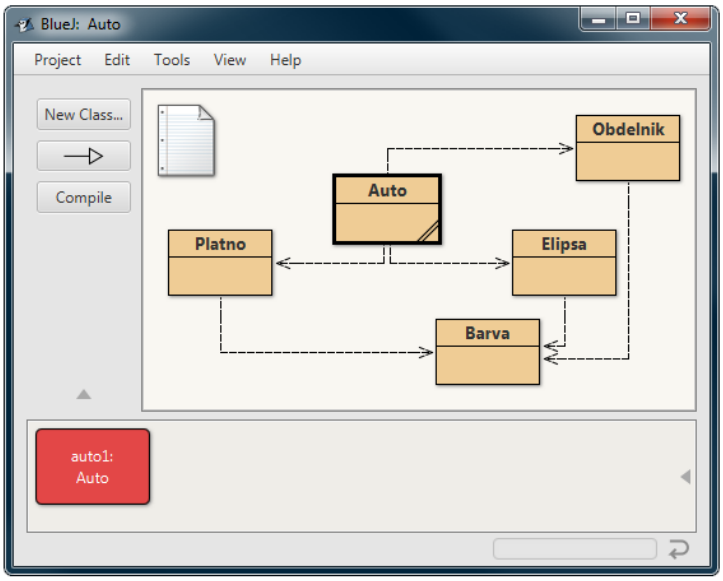
Součástí volby vývojového prostředí můžou být dostupné jazyky v uživatelském rozhraní důležitým faktorem. Někdy se klade ale až moc velký důraz na to, aby s uživatelem komunikovalo vývojové prostředí česky. Na úrovni středního vzdělávání by nemělo být překážkou anglické rozhraní, používáním kterého získá student řadu výhod. Především rozšíří svou slovní zásobu o důležité pojmy, se kterými se obvykle tak či onak bude setkávat. Dále pak anglická rozhraní přináší snadnějšího získání pomoci na Internetu, kde většina návodů, tutoriálů a komunity používá angličtinu. Z těchto důvodů bych se sám zaměřil na fakt, zda vývojové prostředí obsahuje právě jazyk anglický a přítomnost desítek dalších jazykových mutací bych uvítal spíše jako zajímavost.

Obecně platí, že volba vývojového prostředí je pro každého subjektivní a často si vývojáři vystačí i s aplikacemi z kategorie rozšířeného textového editoru jako např. Notepad++ a tak se v praxi žáci setkají s řadou možností. Základem však je vybrat takové prostředí, které jim umožní v začátcích se snadněji zorientovat a později není problém se přesunout na modernější vývojové prostředí.

1.3.1 BlueJ

Původní výukové prostředí a jazyk nazvaný Blue vytvořil v 90. letech německý informatik Michael Kölling jako součást své disertační práce. Ve spolupráci s Johnem Rosenbergem poté v roce 1999 vydali implementaci tohoto prostředí v jazyce Java, což reflektuje název projektu – BlueJ. Od té doby vyvíjel a udržoval aplikaci i díky podpoře tvůrců Javy – Sun Microsystems a později Oracle - tým odborníků sídlící v Austrálii, poté rozšířený do Velké Británie a taktéž Dánska. [9]

BlueJ se přímo zaměřuje svými funkcemi a grafickým rozhraním na výuku programování. Umožňuje studentovi „za běhu“ vytvářet nové objekty, zkoumat jejich stav a vidět graficky znázorněné vztahy mezi jednotlivými třídami a rozhraními. Obsahuje taktéž editor zdrojového kódu se zvýrazněnou syntaxí a méně tradičním visuálním zvýrazněním bloků kódu, což jsou funkce, které začínajícímu programátorovi velmi usnadňují orientaci ve zdrojovém kódu. Zřejmý je taktéž důraz a provázanost s dokumentací ke zdrojovému kódu, kde má student možnost snadno přepínat mezi nimi a vygenerovanou dokumentaci lze taktéž využít k navigaci v rozsáhlém kódu. Na obrázku níže (Obr. 3) lze pozorovat diagram tříd v prostředí BlueJ a taktéž v dolní části panel vytvořených objektů.



Obrázek 3 – Uživatelské rozhraní BlueJ 4.1.3

Na druhou stranu postrádá některé funkce standardní pro jiná vývojová prostředí, která by usnadnila navigaci a to konkrétně přeskakování k definované metodě při kliknutí na část kódu, která ji volá. Taktéž skládání bloků kódu, především těl metod nebo tříd může být často užitečná funkce, kterou v BlueJ postrádám.

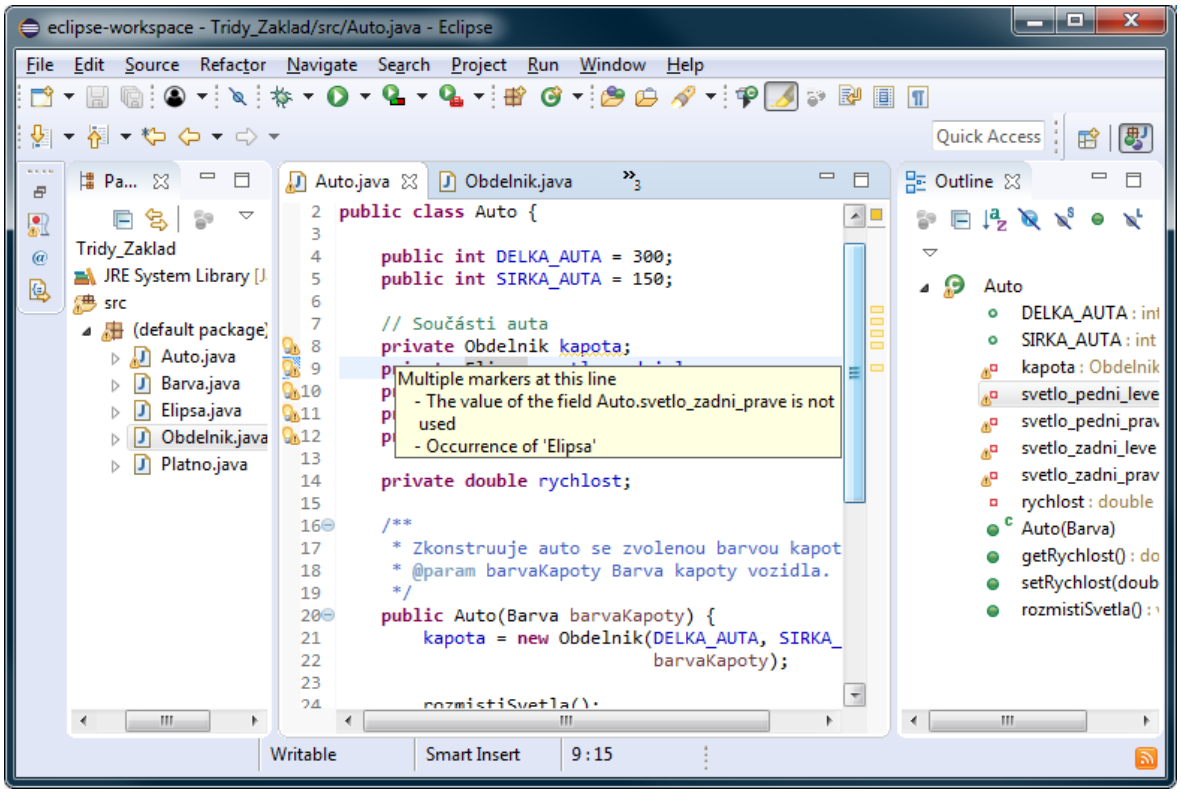
Zkoumaná verze	4.1.2
Operační systém	Windows, Mac OS X, Linux
Licence	GNU GPL v2
Domovský web	www.bluej.org

Tabulka 2 – Vývojové prostředí BlueJ – shrnutí [10]

I přes malé nedostatky nabízí BlueJ skvělé začátečnické prostředí, u něhož není špatnou volbou při programování v Javě začít.

1.3.2 Eclipse IDE

Mezi jedny z nejoblíbenějších vývojových prostředí patří Eclipse vyvíjený neziskovou organizací Eclipse Foundation. [4] Původně vydán na konci roku 2001 je určen především k vývoji aplikací v jazyce Java, avšak za pomocí pluginů dovoluje vývoj i v dalších programovacích jazycích jako jsou např. C, C++, C#, Python nebo Ruby, a v řadě dalších.



Obrázek 4 – Uživatelské rozhraní Eclipse Oxygen.3

Eclipse IDE již obsahuje nástroje pro práci s většími projekty – na obrázku výše (Obr. 4) lze vlevo vidět navigační panel projektu a na pravé straně navigační panel třídy – které v pozdější práci vývojář velmi ocení. U BlueJ jsme viděli diagram tříd, který v případě jednodušších projektů má především demonstrativní účel. U většího množství tříd však

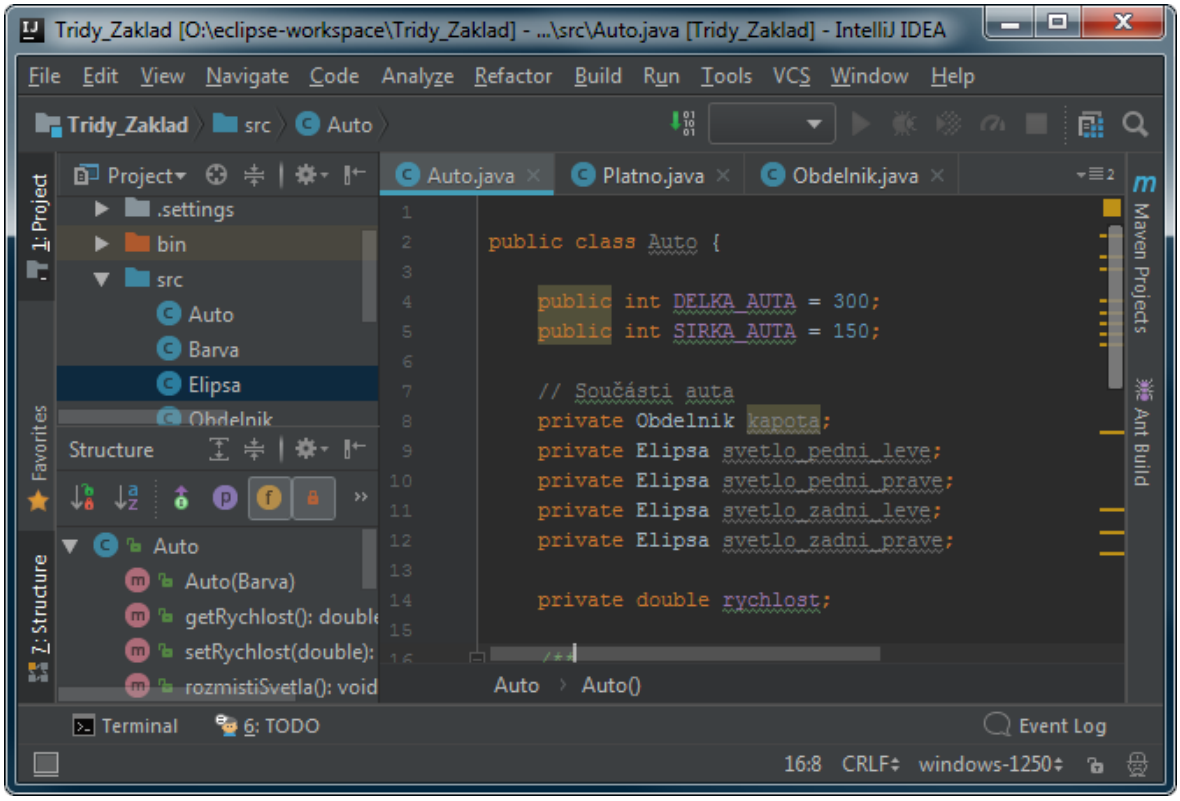
diagram tříd postupně přestane být smysluplný a navigace mezi více balíčky a mezi různými třídami se stane obtížnou. Tyto problémy se snaží, profesionální IDE jako Eclipse, minimalizovat a přecházet mezi jednotlivými třídami nebo metodami nepředstavuje sebemenší problém.

Zkoumaná verze	Oxygen.3 (4.7.3)
Operační systém	Windows, Mac OS X, Linux
Licence	Eclipse Public License (Open Source Software)
Domovský web	www.eclipse.org

Tabulka 3 – Vývojové prostředí Eclipse – shrnutí [11]

1.3.3 IntelliJ IDEA

Další velmi rozšířené vývojové prostředí IntelliJ IDEA zastupuje komerční sféru vývojových prostředí, ačkoliv od verze 9.0 lze aplikaci získat zdarma ve verzi Community Edition. Toto kvalitní IDE, poprvé vydané počátkem roku 2001, obsahovalo jako jedno z prvních pokročilou navigaci ve zdrojovém kódu a taktéž integrovalo schopnosti refactoringu kódu (zlepšení kódu za účelem zpřehlednění se zachováním funkčnosti) a do dnešní doby patří ve vývojových prostředích na špičku. [12]



Obrázek 5 – Uživatelské rozhraní IntelliJ IDEA 2018.1

Na obrázku výše (Obr. 5) lze pozorovat podobné prvky jako u Eclipse IDE, konkrétně projektový panel vlevo nahoře zpřístupňující projektové soubory a použité externí knihovny, dále pak panel struktur obsahující navigaci v atributech a metodách zvolené třídy a jiných součástí a pochopitelně dominantní editor zdrojového kódu. U profesionálních vývojových prostředí se uživatelé obvykle setkají s určitým standardem uživatelských prvků, které jsou pro většinu běžné a uvidíme je i u dalších.

Zkoumaná verze	2018.1 (Community Edition)
Operační systém	Windows, Mac OS X, Linux
Licence	**Community Edition** – Apache 2.0 (Open Source) **Ultimate Edition** – Komerční (proprietární software)
Domovský web	www.jetbrains.com/idea/

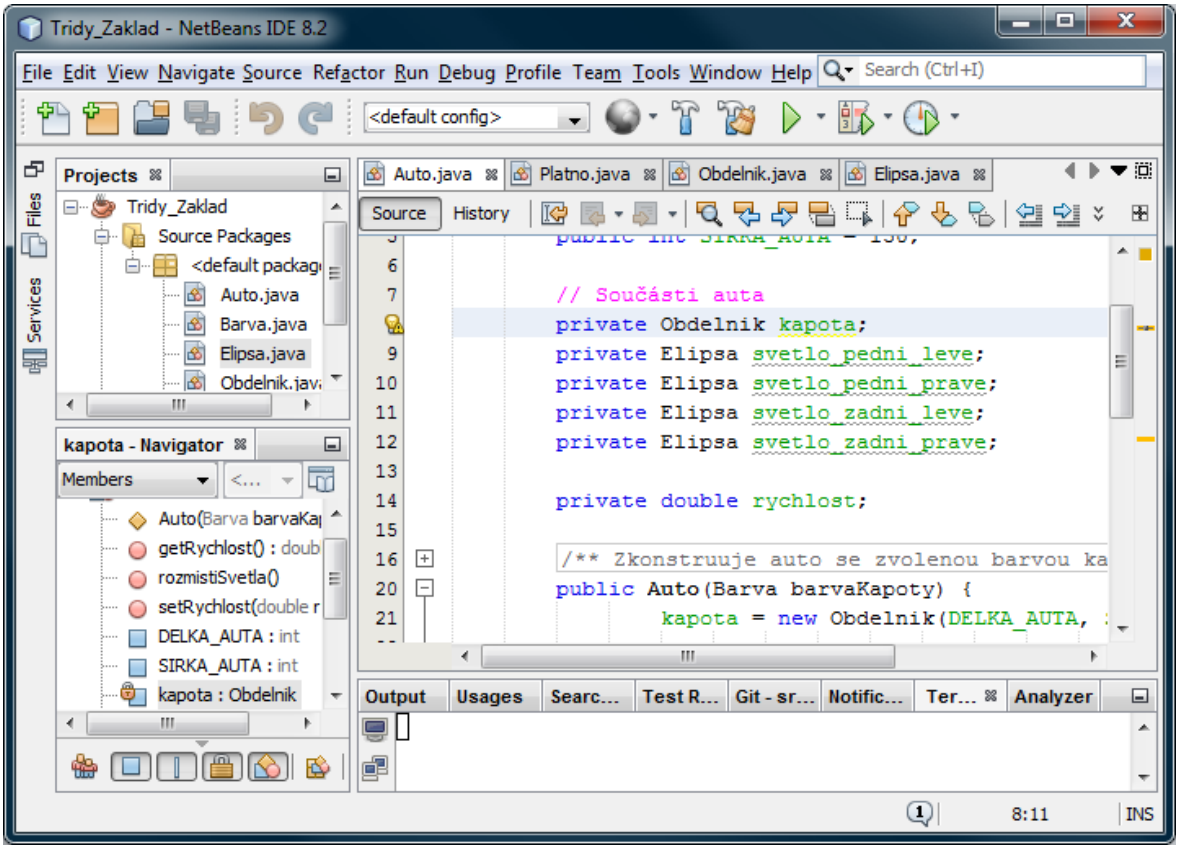
Tabulka 4 – Vývojové prostředí IntelliJ IDEA – shrnutí [13]

Ze zkoumaných vývojových prostředí je IntelliJ IDEA jedné, které aktuálně integruje JShell novou součást Javy z verze 9, která nabízí úžasný potenciál pro výuku Javy, protože umožňuje zapisovat kód v jistých hranicích, aniž bychom museli používat třídy, resp. museli obalovat krátký testovací kód do testovací třídy a hlavní metody. Ale zde se jedná jen o aktuální nedostatek jiných IDE, neboť Java 9 je relativně nová verze a její integrace do různých vývojových prostředí se teprve připravuje.

1.3.4 NetBeans

Posledním zkoumaným vývojovým prostředím jsou NetBeans IDE, původně český zástupce, který vznikl na konci 90. let 20. stol. ze studentského projektu na Univerzitě Karlově v Praze, později zakoupila americká firma Sun Microsystems, vývojář platformy Java, který převedl aplikaci na OpenSource. [14]

Vývojové prostředí se zaměřuje především na vývoj v Javě, ale není mu cizí ani vývoj v jazycích jako PHP, C, C++ nebo webový vývoj v HTML5 nebo JavaScriptu.



Obrázek 6 – Uživatelské rozhraní NetBeans IDE 8.2

V uživatelském rozhraní opět najdeme podobnosti s předchozími vývojovými prostředími avšak funkce, kterou na NetBeans oceňuji je nástroj pro sestavování grafických uživatelských rozhraní, který umožní metodou „drag and drop“ zkompletovat řadu grafických prvků (tlačítka, textová pole, atd.), aniž by se musel uživatel zdržovat se psaním zdrojového kódu, který se automaticky vygeneruje. Díky tomu se může uživatel zaměřit více na funkčnost uživatelského rozhraní své aplikace, než na produkování kódu, který rozhraní sestaví.

Zkoumaná verze	8.2
Operační systém	Windows, Mac OS X, Linux
Licence	Apache Licence (Open Source)
Domovský web	netbeans.org

Tabulka 5 – Vývojové prostředí NetBeans – shrnutí [15]

2 TECHNOLOGIE JAVA

2.1 Historie a verze Javy

Vývoj platformy i jazyka Java započal koncem roku 1990 jako interní projekt americké firmy Sun Microsystems. Původní motivace pro vznik tohoto projektu bylo vytvoření alternativního nástroje k programovacímu jazyku C++, který chtěla firma využít pro vytvoření chytrých spotřebičů nové generace. Tato nová platforma měla řešit problémy se správou paměti, jejíž automatizace osvobozuje vývojáře od zbytečných chyb, které souvisí právě se správou paměti. Další důležitý aspekt představovala přenositelnost mezi různými typy zařízení. Původní záměr byl nový programovací jazyk vycházející z jazyků Mesa a C dále rozšířený do objektově orientovaného prostředí. Po nepodařených pokusech rozšířit samotný jazyk C++ se rozhodli vytvořit jazyk zcela nový, původně pojmenovaný Oak. Později, v roce 1994, se zjistilo, že se jedná o již patentovaný název a jazyk tedy přejmenovali na „Java“. První veřejná beta verze byla dostupná v květnu 1995 a první stabilní verze nazvaná JDK 1.0.2 vyšla v lednu 1996. [16] Od té doby prošla platforma Java velkou řadou změn a updatů.

První velký update nazvaný **JDK 1.1** a vydaný v únoru 1997 se zaměřoval především na použití ve webové oblasti přidáním funkcí, jako jsou:

- **JavaBeans** – objekty určené k serializaci a přenosu,
- **JDBC** (Java Database Connectivity) – rozhraní pro přístup k databázi,
- **Java RMI** (Java Remote Method Invocation) – vzdálené volání metod přes síť.

Součástí tohoto updatu byly i změny ohledně zpracování událostí v grafické knihovně AWT, ale hlavně přidání:

- **vnitřních tříd** do jazyka,
- **JIT** (Just In Time) kompilátoru – současně velmi důležitá součást,
- podpora jazyků ze standardu **Unicode 2.0** – jež nám umožnila ve zdrojovém kódu používat i českou diakritiku. [17]

S novou verzí v září roku 1998 vznikla tzv. „druhá generace“ pojmenovaná **Java 2** (s interním číslem verze 1.2) a celý produkt nesl pojmenování **J2SE 1.2** (Java 2 Standard Edition), aby došlo k odlišení od dalších platforem J2EE (Java 2 Enterprise Edition) a J2ME (Java 2 Platform, Micro Edition). Verze Java 2 tedy přinesla především:

- integraci grafické knihovny **Swing**,
- vybavení **JVM (**virtuálního stroje Java) **JIT** kompilátorem,
- a framework **Collections**. [18]

Jako první významný upgrade nové Javy 2 rozšiřovala verze **J2SE 1.3** vydaná v roce květnu 2000 o další funkce a doplnila stávající funkčnost. [19]

V únoru 2002 následovala verze **J2SE 1.4**, která opět obohatila platformu o nová vylepšení, zejména pak:

- řetězení výjimek (exception chaining),
- podpora protokolu IPv6,
- kanálově založené vstupně výstupní operace,
- a taktéž nové klíčové slovíčko **assert**. [20]

V září 2004 ale přišla doslova revoluce s verzí **J2SE 5.0**. Přínosy této verze byly tak rozsáhlé, že by pojmenování J2SE 1.5, které bylo logicky na řadě dostatečně tuto skutečnost nereflektovalo. Mezi změny přidaných touto verzí patří: [21]

- generické typy,
- automatické zabalování a rozbalování primitivních datových typů,
- výčtový datový typ (nové klíčové slovo **enum**),
- variabilní počet argumentů v metodách (varargs),
- vylepšený cyklus for ve stylu for-each,
- statické importování,
- nebo anotace (taktéž nazývané metadata).

Mnohé z těchto změn samy o sobě by značně změnily styl, jakým se v Javě zapisoval program, ale všechny dohromady způsobily masivní krok vpřed.

Vydáním následující verze **Java SE 6** v prosinci 2006 se opět změnil formát pojmenování produktu. Z „J2SE“ vypadla číslice 2 a název rozšířen na „Java SE“ a číslo aktuální verze. Tato verze dále stavěla na vylepšení z verze J2SE 5.0, upevňovala její funkčnost a bezpečnost a zaměřila se taktéž na optimalizování různých částí platformy. [22]

Koncem ledna 2010 firmu Sun Microsystems získala firma Oracle za částku 7.4 mld. USD. [23]

Další verzi přinesl červenec 2011 a to konkrétně **Java SE 7**, která zahrnuje několik, nejen pro nás, velmi významných vylepšení, kterými jsou hlavně:

- použití textového řetězce (String) uvnitř příkazu **switch,**
- automatickou správu zdrojů ve výjimkách (try with resources),
- automatické odvození datového typu při použití operátoru <> v generických datových typech,
- zjednodušená deklarace variabilního počtu argumentů v deklaraci metod,
- dvojkové literály – zapisování čísla ve dvojkové soustavě,
- použití podtržítka v číselných literálech – pro snazší orientaci v číslech,
- a také vícenásobné zachycování výjimek. [24]

Jedná se o celou řadu praktických užitečných syntaktických vylepšení, které nám opět obohatí sadu nástrojů, které ve zdrojovém kódu můžeme použít.

Jednou z posledních velkých verzí byla v březnu 2014 vydaná verze **Java SE 8**, která znovu změnila způsob programování v Javě tím, že v ní byly představeny tzv. „lambda“ výrazy, které měly za cíl odstranit ve zdrojovém kódu zbytečný a nepotřebný šum a posunout jazyk, spolu s novým Stream API, více směrem k funkčnímu stylu programování. [25] Další významným přínosem bylo přidání JavaScriptového enginu Nashorn, které umožnilo programování v JavaScriptu uvnitř Javy.

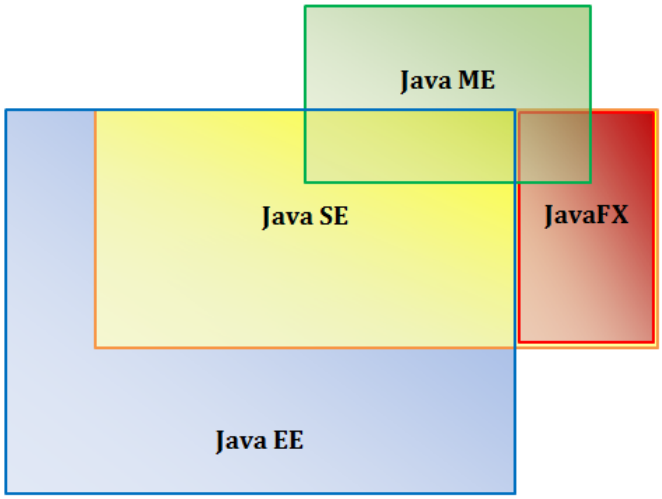
Předposlední dostupnou stabilní verzí Javy je **Java SE 9**, vydanou v září 2017. Většina firem se na přechod na Javu 9 teprve připravuje a stejně tak mnohá vývojová prostředí ji zatím nepodporují, avšak Java 9 obsahuje součást, na kterou lektoři programování, kteří si Javu vybrali, čekali dlouhé roky a tou je Java Shell (taktéž JShell), nástroj, který umožňuje zapsat a spustit Java kód v určité omezené míře tak, že již nepotřebujeme krátký kód obalovat do třídy a metody jen proto, abychom jej mohli spustit. [26] Taktéž masivní změnou je modularizace zdrojového kódu do tzv. „modulů“, které jsou nadřazeny balíčkům. Díky této změně bude možné zabalit aplikaci pouze s knihovnami, které potřebuje a používá a bude takovouto aplikaci možné spustit na zařízení se značně osekanou verzí Javy, která tak na zařízení nebude muset být nainstalovaná kompletně celá.

Poslední verzí, vydanou v březnu 2018, je aktuálně **Java SE 10**, která představila mimo jiné zajímavou novinku v podobě automatického odvození datového typu lokální proměnné pomocí nového speciálního typu „var“. [27]

2.2 Platforma Java

Na rozdíl od jiných kompilovaných (překládaných) programovacích jazyků, se Java drží hesla „write once, run anywhere“, tedy „napsat jednou, spustit kdekoliv“, což v praxi znamená, že program se nepřekládá pro specifickou architekturu procesoru, ale překládá se do tzv. „bytekódu“, jež představuje určitý mezikód, který vykonává Java Virtual Machine (JVM), tedy software, který na základě bytekódu nechá vykonat příslušné instrukce platformu operačního systému, na němž je virtuální stroj spuštěn. Tento postup zajišťuje, že programátor nemusí věnovat tak velkou (někdy žádnou) pozornost tomu, pro který operační systém a procesor aplikací vyvíjí. Jediným požadavkem na daný systém pak je, aby obsahoval Java Runtime Environment (JRE), který obsahuje příslušný virtuální stroj pro Javu.

Pro účely vývoje aplikací musí ovšem vývojář či student mít širší paletu nástrojů pro vývoj, která se nazývá Java Development Kit (JDK). Tato vývojářská sada obsahuje především Rodina platform Java se ale skládá z několika platforem, které slouží různým účelům.



Obrázek 7 – Diagram vybraných platforem Java [28]

Java Standard Edition (Java SE) slouží k vývoji desktopových, potažmo serverových aplikací a pro účely výuky je to výchozí platforma, která obsahuje veškeré potřebné nástroje. **Java Enterprise Edition (Java EE)** rozšiřuje Java SE se zaměřením na distribuované aplikace, a taktéž poskytující funkce ke zpracování transakcí tak, aby výsledná aplikace byla bezpečná a škálovatelná. [29] **Java Platform Micro Edition (Java ME)** poskytuje nástroje určené k vývoji pro mobilní zařízení (především chytré telefony) ale vypůjčuje si některé knihovny i z Java SE. V neposlední řadě pak **JavaFX** nabízí nástroje pro obohacení především webových aplikací o interaktivní prvky pro komunikaci s uživatelem.

3 DOSTUPNÉ VÝUKOVÉ MATERIÁLY

Java, jako přinejmenším jeden z nejpopulárnějších současných programovacích jazyků, se těší silné podpoře velké komunity, která okolo ní od jejího počátku vznikla. Řada webů obsahuje sekce věnující se Javě, o vývoji na platformě Java byla vydaná dlouhá řada knih, ale to automaticky neznamená, že se jedná o vhodné zdroje informací do školního prostředí.

3.1 Elektronicky dostupné zdroje

Oficiální dokumentace Java Platform SE & JDK – považuji za nejdůležitější zdroj, především proto, že je ze všech zdrojů nejaktuálnější a studenti by o existenci oficiální dokumentace měli vědět a měli by být schopni s ní pracovat. [30]

The Java Tutorials – rozsáhlá databáze návodů k nejrůznějším aspektům Javy od začátečnických až po rozhraní pro profesionální programátory. [31]

algoritmy.net – web věnující se, jak název napovídá, algoritmům, zejména řadícím nebo vyhledávacím algoritmům, ale zaobírá se i dalšími relevantními tématy jakými jsou datové struktury, teorie algoritmů nebo automaty a gramatiky. [32]

Tento web zpracovává zvolenou tématiku velmi přehledně a často doporučuje i literární zdroje, ze kterých čerpal a kde lze nalézt další informace.

itnetwork.cz – web, obsahující tutoriály a ukázkové programy pro celou řadu programovacích jazyků jako např. C, C++, Python, PHP, JavaScript nebo právě Java. [33]

Web se věnuje především tomu, jak daný jazyk používat a to na konkrétních příkladech. V sekci Java se kurzy obvykle dělí na výkladovou část, která je zdarma a cvičení, která jsou ze začátku kurzu zdarma a další cvičení jsou již prémiové, k nimž lze získat přístup za finanční poplatek nebo přispěním vlastního článku na web.

vyuka.pecinovsky.cz – webové stránky Ing. Rudolfa Pecinovského, CSc., jednoho z nejznámějších a nejvýznamnějších odborníků na výuku programování. Stránky obsahují jeho přednáškové prezentace i videozáznamy přednášek a taktéž výukové projekty. Jak se zde uvádí, daný web slouží jako rozcestí, které člověka zavede i k jeho četným publikacím. [34]

3.2 Výuková literatura

Ve světovém měřítku vychází publikací týkající se programovacího jazyka a vývoje na platformě Java velká řada. Naprosto běžným jevem je, že při každém vydání nové verze platformy Java jsou vydány autory publikací rozšířená vydání jejich knih. Rozdělil jsem proto přehled na zahraniční a tuzemské a vynechat pokud možno starší edice. Přehled tedy začíná zahraničními publikacemi:

Effective Java (Bloch, 2018) – je oblíbená publikace pro pokročilé vysvětlující prvky Javy včetně těch nejaktuálnějších z pohledu profesionálního vývojáře a univerzitního profesora v jedné osobě. [35]

Java 8 in action: lambdas, streams, and functional-style Programming (Urma, Fusco a Mycroft, 2015) – zahraniční publikace zabývající se především efektivním využitím Javy 8 za využití nových funkční rozhraní a lambda funkcí. [36]

Java: The Complete Reference (Schildt, 2017) – Velmi rozsáhlá publikace pokrývající velkou část jazyka Java aktualizovaná pro verzi 8, jeho syntaxi, všechny důležité knihovny i vytváření uživatelského rozhraní. [37]

Pak také existuje mnoho souvisejících publikací od českých autorů. Mezi vybrané patří (řazeno chronologicky):

Úvod do Javy (Pavlíčková, Pavlíček, 2005) – Tato skripta určená hlavně pro studenty obsahují nejdůležitější okruhy pro výuku Javy tak, aby čtenářům usnadnila počátky. [38]

Algoritmy a programovací techniky (Töpfer, 2007) – Učebnice, zaměřující se na obecné postupy, algoritmy a techniky v programování a základní datové struktury. [39]

Java: grafické uživatelské prostředí a čeština (Herout, 2007) – kniha navazující na autorovu předchozí publikaci Učebnice jazyka Java se věnuje tvorbě uživatelského prostředí za použití knihoven AWT a Swing a taktéž problémů spojených s použitím české diakritiky a kódování v grafickém rozhraní vytvářené aplikace. [40]

Java 8: úvod do objektové architektury pro mírně pokročilé (Pecinovský, 2014) - nejnovější publikace výše zmíněného Rudolfa Pecinovského rozšiřující jeho předchozí knihu o novou funkčnost z Javy 8. [41]

II. PRAKTICKÁ ČÁST

4 CÍLE A ZVOLENÉ METODY PRAKTICKÉ ČÁSTI

4.1 Vyučování programování

S rostoucí popularitou objektově orientovaného programování je poptávka po jeho výuce velká a v poslední dekádě se stala prakticky nedílnou součástí výuky program. Jako dědictví minulosti jsme při výuce vycházeli z toho, že se žáci nejprve naučí strukturované programování, algoritmy a datové struktury a objekty se naučí jako pokročilejší nástavbu na datové struktury později. Při volbě vyšších programovacích jazyků jakými jsou Java nebo C++ však hned na začátku narazí vyučující na problém. Tradiční výuka začíná tím, že se žákovi prezentuje nějaký jednoduchý kousek kódu a demonstruje se jeho funkčnost. Často to bývá programátorům známý „Hello World“ program, jehož cílem je na standardní výstup vypsat dvě výše zmíněná slova. Pokud ale tento program chceme zapsat v námi zvolené Javě, vznikne z našeho záměru 5 následujících řádků kódu:

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

Pokud žákovi těchto pět řádek kódu prezentujeme, můžeme očekávat lavinu otázek, zjišťujících, co znamená public, class, static, void, main, String, args, System, out, println, ačkoliv nás zde zajímá pouze tučně zvýrazněný řádek. Zde budeme hned od začátku nucení učinit volbu, zda mu sdělíme obligátní odůvodnění, že se to dozví později v semestru anebo se uchýlíme k tomu, že dané konstrukce vysvětlíme.

Kölling uvádí, že je mnohem obtížnější naučit žáka objektově orientovanému programování poté, co již byl vyučován ve směru programování strukturovaného, než vyučovat objektově orientované paradigma jako první. Taktéž uvádí, že není možné vyučovat objekty jako pouhou nástavbu nad strukturami, neboť přimýšlení s objekty je diametrálně odlišné. [42]

Potenciální výhoda objektově orientovaného přístupu je i ta, že ji lze vyučovat na abstraktní úrovni i mnohem dřív (např. na úrovni základní školy). Kde je možné využít specializovaných aplikací, které umožní žákům pozorovat výsledky jejich práce, což i v programování, řekl bych, podstatná část učení. Motivace tedy začít výuku objektově orientovaným přístupem a až později se zaměřit na strukturované programování je tedy jasná.

4.2 Rozsah a tematické okruhy

Při volbě tematických okruhů je podstatné věnovat pozornost tomu, jaké množství látky je možné za dostupný čas probrat. Vývoj aplikací na platformě Java (ale i obecně) je velmi široká oblast. Z toho důvodu byly vybrány ty nejpodstatnější témata, které je nutné pro uvedení studentů do programování v Javě probrat.

Název tématu	Vyučovací cíle
Základy OOP	• Vysvětlit základní pojmy OOP (třída, instance, objekt, zpráva, metoda, návratová hodnota) • Uvést studenty do vývojového prostředí BlueJ • Umožnit žákům vyzkoušet si manipulaci s objekty a jejich interakci.
Základní syntaxe a vlastní třída (atributy)	• Vysvětlit základní syntaxi, výraz • Vysvětlit žákům strukturu třídy, její deklaraci a definici zapsané v jazyku Java • Objasnit deklaraci a definici atributů
Vlastní třída (metody)	• Vysvětlit význam a zápis konstruktorů • Popsat smysl přetěžování metod a konstruktorů
Atributy třídy a objektu	• Objasnit atributy a metody na úrovni třídy a instance a jejich zápis
Primitivní datové typy	• Uvést studenty do primitivních datových typů a jejich aritmetiky
Podmínky	• Probrat se studenty více do hloubky tok provádění operací a nutnost jeho řízení • Představit studentům konstrukci podmínky a logického datového typu • Vysvětlit studentům rozsah platnosti proměnných
Výčet a String	• Probrat základy výčtového datového typu a třídy String
Balíčky a přístupové modifikátory	• Osvětlit existenci balíčků • Objasnit princip přístupových modifikátorů
Rozhraní	• Vysvětlit smysl a použití pro rozhraní • Ukázat a vyzkoušet deklaraci a implementaci rozhraní
Dědičnost	• Objasnit princip dědičnosti a jeho použití v OOP

Tabulka 6 – Téma a výukové cíle

4.3 Role materiálů ve výuce

Práce učitele programování je, řekl bych, dynamický proces, který by se měl přizpůsobovat zejména v horních limitech schopnostem žák, ačkoliv je zřejmé každý ze studentů bude mít rozdílnou výchozí pozici. Vytvořené materiály tak může využít jako opěrné body pro velké tematické celky, jež nastiňují.

5 VYŽADOVANÉ SOFRWAROVÉ PROSTŘEDKY

5.1 BlueJ

5.1.1 Požadavky a instalace

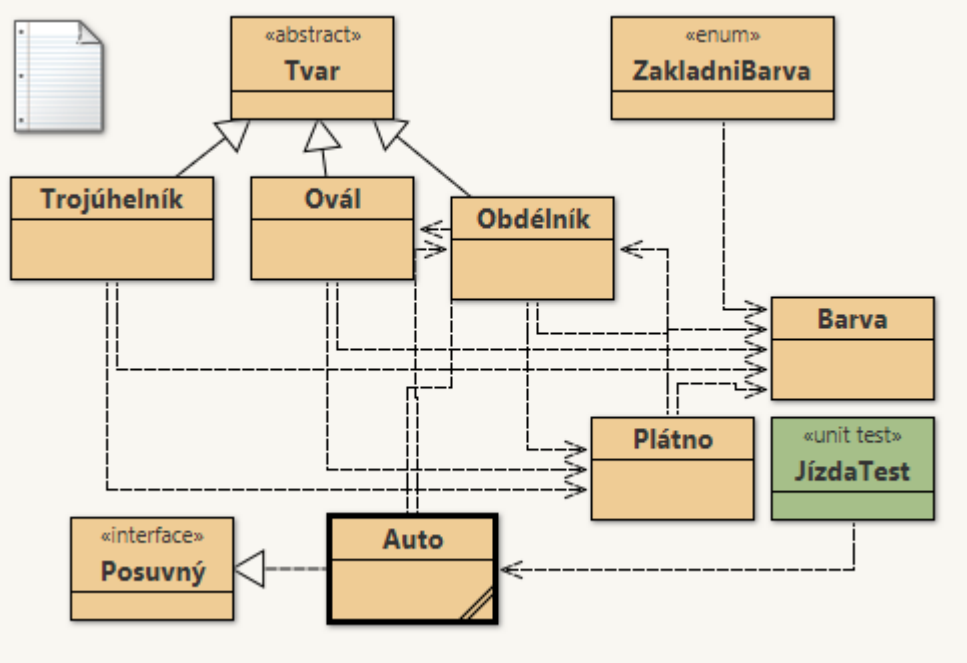
Aktuální verzi BlueJ získáme na oficiálních webových stránkách www.bluej.org, kde jsou v nabídce současné verze pro běžné operační systémy. Vývojové prostředí BlueJ, které je zde popsáno, je v aktuálně nejvyšší verzi 4.1.2, jež vyžaduje pro spuštění minimální verzi operačního systému Windows 7. Pokud bychom chtěli snad nainstalovat BlueJ na nižší verzi operačního systému Windows jako např. Windows XP, musíme použít verzi 3.1.7. V případě kontroly kompatibility s ostatními staršími operačními systémy jsou na tomtéž webu pod odkazem „*Download previous versions or source code*“ všechny předešlé verze BlueJ. Pokud nejsou k dispozici práva pro instalaci, může být praktickou volbou i spouštění pomocí tzv. „portable aplikace“, tedy verze BlueJ, kterou lze bez instalace spustit kupříkladu z USB flash disku.

Pozitivním aspektem instalace nových, ale i některých starších, verzí BlueJ je, že není nutné zajišťovat instalaci JDK (Java Development Kit) neboť je obvykle součástí instalace samotného vývojového prostředí a případnou nainstalovanou verzi JDK neovlivňuje. Taktéž od verze 3.0.9 lze snadno pomocí dialogu Preferences a záložky Interface (tedy z výchozí anglické lokalizace) nastavit jazyk uživatelského rozhraní z široké škály včetně češtiny.

Instalace BlueJ probíhá velmi rychle, v 5 krocích se zvolí pouze základní a pro každou instalaci standardní informace jako umístění, rozmístění nastavení (instalace pouze pro jednoho nebo pro všechny uživatele) a taktéž rozložení zástupců (na plochu nebo do nabídky Start).

5.1.2 Uživatelské prostředí

Po instalaci a spuštění BlueJ se mohou žáci seznámit s uživatelským rozhraním. Toto rozhraní se skládá ze dvou stěžejních částí a těmi jsou diagram tříd, který tvoří dominantní část rozhraní, a lišta objektů, kde uživatel najde odkazy na vytvořené objekty.



Obrázek 8 – Diagram tříd v BlueJ

V diagramu tříd jsou jednotlivé součásti, tedy třídy, abstraktní třídy, rozhraní, výčtové typy enum nebo testovací třídy, zobrazeny jako obdélníky s jejich názvem napsaným tučně, kde jejich typ je dále specifikován buď specifikujícím textem (např. «interface» pro rozhraní) nebo barvou (UnitTest má odlišnou barvu od standardních prvků). Bílý list vlevo nahoře odkazuje na soubor *README.TXT*, do kterého si žák může zapisovat úkoly a poznámky k danému projektu.

Přerušované čáry s šipkou v diagramu ukazují na prvky, které daný prvek v sobě využívá. Např. z diagramu výše lze vyčíst, že třída Auto využívá tříd Ovál a Obdélník. Dalším druhem čáry v diagramu tříd je plná čára s trojúhelníkovou šipkou, která znázorňuje dědičnost, kde třída, na kterou šipka ukazuje, je rodičem třídy, ze které čára vychází.

S jednoduchým diagramem tříd, lze znázornění těchto vztahů využít pro lepší přehled mezi jednotlivými třídami. Nevýhodou diagramu tříd v BlueJ je, že se jen velmi obtížně přizpůsobuje představám o tom, jak by si uživatel spojnice mezi třídami nakreslil sám, neboť s rostoucí složitostí projektu, kde se třídy vzájemně hojně využívají, není snadné se ve vel-

kém množství šipek a čar vyznat. Prvky v diagramu lze pomocí myši volně posouvat a měnit jejich velikost s cílem dosáhnout optimálního rozložení. Bohužel bez možnosti lépe ovládat obtékání šipek, je často velmi obtížné umístit třídy v diagramu dostatečně přehledně.

V začátcích práce s BlueJ bych považoval za vhodné doporučit žákům, ať si mezi jednotlivými hodinami projekt pravidelně zálohují, aby nepřišli o svou práci a své realizované nápady, kdyby se mělo cokoliv stát.

Obrázek 9 – Lišta objektů v BlueJ

Lišta objektů představuje další významnou část uživatelského rozhraní a umožní nám získávat odkazy na instance tříd, které jsou nám dostupné. Odkazy poté můžeme využít při zadávání parametrů metod nebo pro jejich inspekci, tedy nahlížení do útrob objektu a v jakém stavu jsou hodnoty atributů.

Toto uživatelské rozhraní celkově umožní žákům vytvářet instance tříd a prozkoumávat stav jednotlivých objektů promptnějším způsobem, než je běžné, což jim v začátcích pomůže lépe se zorientovat.

5.2 REPL a JShell

Read-Eval-Print Loop (zkráceně REPL) je v obecném slova smyslu programovací rozhraní, které umožňuje vyhodnocování uživatelských vstupů ve formě výrazů. S vydáním Javy 9 jsme dostali k dispozici již zmíněný JShell, tedy verzi REPL rozhraní pro Javu, které nám při vyučování umožní oprostit se od zbytečných obalových tříd a metod, pokud máme v úmyslu se zabývat výukou strukturovaného programování a jednoduchých algoritmů. Prozatím se jedná o relativně novou věc, kterou mnohá vývojová prostředí v budoucnu teprve implementují. Za aktuálně nejpřívětivější implementaci bych považoval tu v IntelliJ IDEA, která dovoluje psát, ukládat a spouštět přímo úryvky kódu snadno a přímo v aplikaci. JShell se totiž primárně spouští pomocí příkazové řádky, kde jej lze pomocí standardního vstupu ovládat a vkládat do něj příkazy nebo zdrojový kód. Takováto manipulace ale má svá specifika, které můžou a nemusí být zrovna dle představ toho, jak chceme žákům algoritmy přiblížit.

JShell nabízí k práci mnoho běžně používaných balíčků a tříd, které se při práci s ním automaticky importují. Ať už se vyučující rozhodne vyučovat strukturované programování jako první nebo ne, pomůže mu JShell oprostit studenty od šumu v podobě tříd a metod, které se proberou až později. I v opačném případě, kdy již studenti objektům rozumí, není dle mého názoru zlé mít po ruce nástroj, který při probírání algoritmů očistí zdrojový od konstrukcí, které nejsou pro algoritmus podstatné.

Podstatné je také zmínit, že svou verzi REPL rozhraní implementuje i BlueJ prostřednictvím svého Příkazového panelu (angl. Code pad), které lze podobným způsobem jako JShell využít s tím, že je více integrovaný do BlueJ (umožňuje např. přebírat odkazy na objekty), ale prozatím nespoléhá na Javu 9. Vkládání delších příkazů do příkazového panelu je taktéž poněkud obtížné, neboť je k tomu k dispozici pouze jednořádkové vstupní pole. Vkládat tedy určitou část kódu pro otestování algoritmu není úplně vhodné, ačkoliv to v některých případech není nemožné. Osobně bych ale preferoval, kdyby v BlueJ existovala konstrukce kousku Java kódu, který by byl buď prostřednictvím příkazového panelu nebo v budoucnosti integrovaným JShellem spustitelný samostatně. V každém případě je přítomnost rozhraní Příkazového panelu v BlueJ pozitivní funkce, protože umožňuje provádět ad-hoc výpočty, které lze použít pro vytvořené objekty.

6 PODPŮRNÉ MATERIÁLY PRO VÝUKU

6.1 Základy objektově orientovaného programování

Objektově orientované programování se snaží přistupovat k programování způsobem, kde existují pouze objekty, které mají určité pasivní vlastnosti, ale také mají schopnosti, prostřednictvím kterých mezi sebou nebo i jednostranně komunikují.

6.1.1 Základní pojmy OOP

Jedením z nejzákladnějších pojmů OOP je třída, jež představuje formu nebo jakýsi prototyp, pro konstruovaný objekt. Jejím základním cílem je popisovat objekty, které mají společné vlastnosti a schopnosti.

Příkladem může být řekněme třída Auto, která zachycuje aspekty, která mají auta společná. Jedním z takových aspektů může být stav paliva, rychlost nebo počet dveří. Různá auta mají konkrétní hodnotu těchto vlastnosti různá, ale obecně je všechna auta sdílí. Instance třídy nebo obecně objekt je už konkrétní realizací určité třídy, například již zmíněného auta. [43]

Aby svět objektů nebyl statický a nečinný, musí být objekty schopny mezi sebou komunikovat. Tato komunikace se uskutečňuje prostřednictvím tzv. „zpráv“. Zprávy mohou být bez odezvy (tedy jednosměrné) nebo s odezvou (tedy s návratovou hodnotou). Za zprávy lze považovat nejrůznější interakce, v extrémech i na molekulární úrovni. Když člověk otevírá dveře, tak jeho ruce vysílají zprávu dveřím, ať se začnou otevírat. Dveře, podle toho, zda jsou fyzicky odemknuty a mají prostor kam se otevřít, zareagují. Celou interakci zde řídí fyzikální zákony, ale ať už je řídí ony nebo na vyšší úrovni počítačový systém, princip je stejný.

6.1.2 Vybrané koncepty OOP

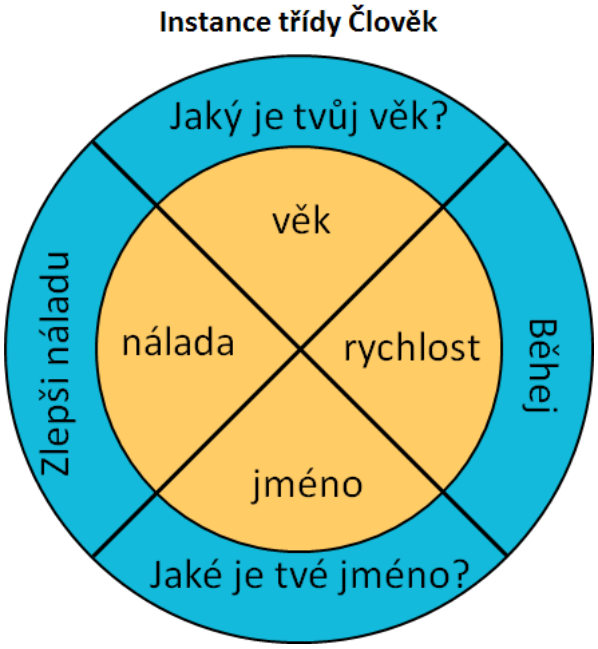
Předtím než se studenty začneme probírat, jakým způsobem prakticky manipulujeme a později i vytváříme třídy a instance, je důležité představit čtyři základní koncepty objektově orientovaného programování. Během kurzu pak studentům demonstrujeme prakticky aplikaci těchto konceptů a zásad.

Abstrakce je jedním ze základních kamenů nejen objektově orientovaného programování, ale i návrhu softwaru obecně. Na základě něj programujeme struktury tak abstraktní, aby je bylo možné využít pro více než jeden účel. Příkladem takového postupu je např. metoda,

která řadí položky podle určitého kritéria. Taková metoda je navržena obvykle obecně tak, aby byla schopna seřadit jednorozměrnou řadu jakýchkoliv prvků, pokud kromě vstupních položek k seřazení obdrží i pravidlo, na základě kterého dokáže rozhodnou, zda je jeden prvek větší než druhý. Dalším příkladem může být struktura dynamického pole, které aniž by předem vědělo, jaký druh a kolik prvků v něm bude uloženo, se při jeho použití dokáže patřičně přizpůsobit. [44]

Ten se zakládá na principu, že procedury, struktury, objekty a různé další části systému, se kterými softwarový inženýr pracuje, můžou být různě složité

Zapouzdření patří mezi podstatné koncepty OOP, neboť objekty ve své podstatě chrání. Základní myšlenka zapouzdření spočívá v tom, že vnitřní stav objektů lze zjistit nebo ovlivnit pouze za pomocí metod objektu, které určují, jak se to má dít.



Obrázek 10 – Diagram instance Člověka

Mějme objekt člověka popsaný tímto diagramem (Obr. 10). Daný člověk má 4 atributy (věk, rychlost, jméno a náladu). Tyto atributy ale nelze přímo zjistit ani nastavit. Člověk definuje 4 konkrétní metody, které mohou být pro přístup použity. 2 metody zjišťují hodnoty atributů (v diagramu formou otázky, které je možné položit) a 2 metody mění hodnoty atributů (v diagramu formou rozkazu). Tyto metody můžou mít i parametry, které případné rozhodování, zda sdělit nebo změnit hodnoty atributů, mohou usnadnit.

Dědičnost a Polymorfismus jsou jako další důležité koncepty OOP uvedeny níže v příslušných vyučovacích hodinách.

Po krátkém úvodu do základních pojmů OOP je třeba seznámit žáky s vývojovým prostředím. V přiložených materiálech (složka *2_Základy_OOP*) obsahuje projekt BlueJ s hrstkou tříd, kde je možné využít pro vyzkoušení si manipulace s objekty, jejich vytvářením a voláním metod. Součástí materiálů je taktéž krátká prezentace k instalaci prostředí BlueJ a základní orientaci, ale nejlépe se žáci zorientují tím, že si manipulaci sami vyzkouší.

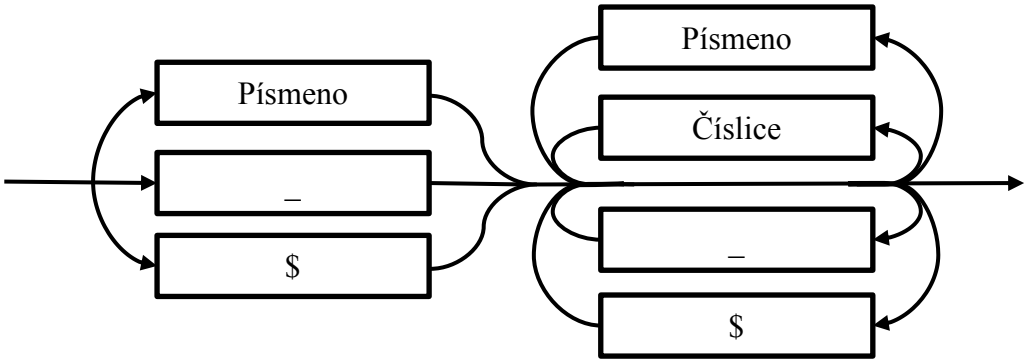
Aby mohl snadněji ukázat, že dovedou vyvolat posloupnost příkazů, které vedou k určitému cíli, dostanou za úkol pro tento účel využít speciálního zabudovaného druhu třídy, jímž je UnitTest. S jeho pomocí může zaznamenat přesně, které příkazy vykonal nebo které objekty vytvořil a následně tuto posloupnost příkazů dvěma kliky myši zopakovat, aniž by během toho procesu musel napsat jediný řádek kódu. Tato metoda a celkově princip manipulace s objekty v prostředí BlueJ umožní studentovi se snadněji a lépe sžít s vytvářením a manipulací s objekty.

6.2 Základní syntaxe a vlastní třída (atributy)

Syntaxe jazyka Java je velmi podobná jazykům z rodiny C, ale je potřeba zdůraznit, že se jedná o naprosto jiný jazyk a syntaxe je jedna z nemnoha věcí, které tyto jazyky spojují. Syntaxe je ovšem celkem přívětivá a s jejím pochopením by neměl mít problémy ani začátečník.

6.2.1 Identifikátory

Abychom mohli pojmenovávat v Javě proměnné, třídy a další prvky, musíme dodržovat určitá pravidla a měli bychom dodržovat určité konvence. Takovému názvu se obecně říká identifikátor (identifier) a pravidla pro jejich tvoření by se dala shrnout následujícím diagramem:



Obrázek 11 – Tvorba identifikátoru

Konkrétně tedy prvním znakem identifikátoru může být písmeno, podtržítko, případně znak dolaru. Jakýmkoliv následujícím znakem můžou být dříve zmíněné, ale taktéž číslice.

Minimální délka identifikátoru je 1 znak, maximální délku samotná specifikace Javy neomezuje. V praxi však platí, že by názvy měly být krátké, ale dostatečně deskriptivní na to, aby nebylo nutné jejich význam, pokud možno vůbec, vysvětlovat. Pokud tedy zapisujeme krátkou metodu pro výpočet obsahu obdélníku, dalo by se přijmout, že jsou proměnné pro délky jednotlivých stran pojmenovány **„a**“ a „**b**“, neboť bude poměrně zřejmé, jaký je jejich význam. Pojmenovat tyto proměnné „**stranaA**“ a „**stranaB**“ je taktéž přijatelné a dost možná preferovaná, pojmeme-li podezření, že by první možnost mohla způsobit nejasnosti, avšak pojmenovávat proměnné „**x1**“ a „**x2**“ by mohlo v tomto případě čtenáře zmást.

Co se znaku dolaru týče, ačkoliv je jeho použití povoleno, doporučuje se jej nepoužívat vůbec a v případě podtržítka se nedoporučuje jeho použití jako prvního znaku identifikátoru.

Důležitým aspektem vytváření identifikátorů je fakt, že Java rozlišuje velikost písmene (je tzv. „case-sensitive), což umožňuje přijmout řadu konvencí. Tyto konvence jsou uvedeny u příslušných sekcí, ale již z počátku lze uvést, že obvykle volíme převážně malá písmena, ale pro lepší čitelnost se používá tzv. „velbloudí notace“ (angl. Camel Case). Ta říká, že při víceslovném identifikátoru se začíná buď velkým, nebo malým písmenem (dle zvolené konvence) a počáteční písmeno každého dalšího slova je velké, například:

```
obsahCtverce
barvaPozadi
```

Ačkoliv výše uvedené identifikátory jsou bez čárek a háčků, používat v identifikátorech diakritiku je povolené a z počátku nejspíš mnohem přirozenější, neboť to v určitých situacích je nevyhnutelné, aby nedošlo ke špatnému pojmenování. Z tohoto důvodu někdo preferuje použití angličtiny při pojmenovávání proměnných a metod. Zde záleží na programátorově subjektivní znalosti angličtiny.

Další limitací při volbě identifikátoru jsou klíčová slova a 3 literály (true, false a null), jež splňují pravidla a omezení pro identifikátory, ale jazyk si je vyhrazuje a přiřazuje jim speciální význam. Použití klíčových slov na místě identifikátoru si žák hlídat sám nemusí, neboť jsou takováto slova v editoru obvykle zvýrazněna, na jejich špatné použití je upozorněno vývojovým prostředím již při psaní, a pokud by obě tato varování programátor ignoroval, tak se výsledný zdrojový kód ani tak nepřeloží do spustitelného programu a překladač zdůvodní, proč se tak stalo.

6.2.2 Deklarace třídy

Po experimentování s objekty a manipulaci s nimi je vhodná doba začít vytvářet třídy vlastní. Kliknutím pravým tlačítkem myši do volného prostoru v diagramu tříd a zvolením položky Nová třída (případně pomocí menu Úpravy a položky Nová třída) lze vytvořit novou třídu zvoleného typu. Pro začátek žáci vytvoří „Standardní třídu“, BlueJ vytvoří třídu se zadanými parametry a vyplní zakomentovanou kompletní strukturou třídy. V tuto chvíli je vhodné danou strukturu smazat a ponechat pouze holou deklaraci třídy. Ačkoliv předepsaná struktura má svůj smysl, jedná se v začátcích o velké množství šumu, které

musí žáci zbytečně vnímat a je mnohem vhodnější udržovat kód pokud možno jednoduchý a přehledný. Nejzákladnější deklarace třídy má podobu [45]:

```
class NázevTřídy {
    // atributy
    // konstruktory
    // metody
}
```

Již na tomto úryvku lze upozornit na červeně zvýrazněné klíčové slovo *class*, které indikuje deklaraci třídy a je jedno z prvních klíčových slov, se kterým se žáci v kurzu setkávají. V souvislosti s konvencemi identifikátorů zmiňovaných výše je třeba zdůraznit, že třídy se konvenčně pojmenovávají již uvedenou „velbloudí notací“ avšak ve variantě s velkým počátečním písmenem. Za název třídy volíme z pravidla podstatná jména, která by stejně jako ostatní identifikátory měla být krátká, ale výstižná a doporučuje se, aby byla nejméně 2 znaky dlouhá, neboť jeden znak dlouhé typové identifikátory si konvence vyhrazuje pro účely generických typů.

Výše uvedená deklarace je pouze ta nejjednodušší a obsahuje minimum vyžadovaného pro deklaraci třídy. Hlavička třídy – zde složená pouze z klíčového slova class a názvu třídy – může být rozšířena o další informace, které však z počátku není třeba uvádět, neboť by to jen komplikovalo pochopení žáků. Tyto upřesňující deklarace je vhodné probrat až s příslušným učivem (konkrétně tedy přístupové modifikátory, další modifikátory, rozhraní a dědičnost, v případě zájmu i generické typy).

6.2.3 Atributy objektu

Předtím než začneme ve třídě používat atributy objektu, musíme nejprve deklarovat jejich datový typ a název, případně další modifikátory. Základní deklarace tedy vypadá takto:

```
datovýTyp názevProměnné;
```

Chci-li tedy deklarovat celočíselnou proměnnou uchovávající věk, a desetinné číslo uchovávající váhu zapíšu jednoduše [46]:

```
int věk;
double váha;
```

Již nyní můžeme začít zdůrazňovat, že všechny příkazy (tedy i deklarace proměnných) musíme zakončit středníkem. Středníky jsou se začátky programování velká kapitola, protože studenti na ně budou pravděpodobně velmi často zapomínat a, ačkoliv je na tuto chybu vývojové prostředí nebo samotný překladač upozorní, jsou středníky obvyklým zdrojem

frustrace. Datových typů, pomocí kterých lze deklarovat proměnnou je několik, ale v úvodu postačí uvést 2 základní kategorie:

1. Primitivní datové typy
2. Referenční datové typy

Ačkoliv je primitivních datových typů celkem 8, neuváděl bych v začátcích jejich celý výčet. Naopak bych uvedl jen ty nejdůležitější zástupce, kterými jsou *int*, *double* nebo *boolean*, a zaměřil se na práci s prvními dvěma jmenovanými, protože s čísly se žákům pracuje nejsnadněji, neboť operace na nich prováděné již znají z matematiky.

Referenční datové typy lze pochopit naproti tomu trochu lépe, protože je můžeme demonstrovat na mnohem konkrétnějších příkladech. Zároveň také bychom měli upozornit, že všechny třídy včetně těch, které vytvoříme my, se automaticky stanou novými datovými typy, které lze použít jak v jiných třídách, tak v daných třídách samotných.

Při deklaraci atributu v třídě a obecně jakékoliv proměnné je nedílnou součástí i proces určování hodnoty dané proměnné. Tento proces může mít dvě podoby, jimiž jsou:

1. Automatická inicializace
2. Přiřazovací příkaz

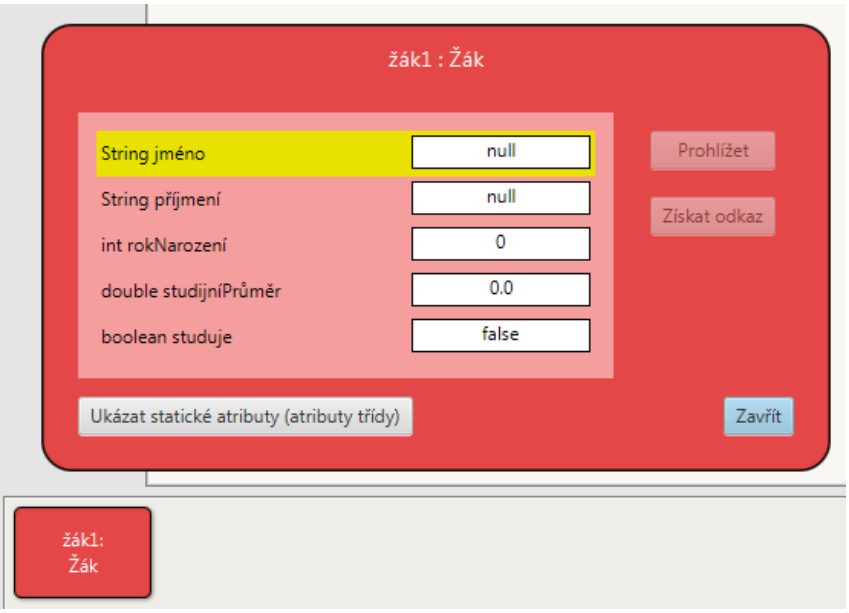
Jako první uvedu automatickou inicializaci, na kterou se později často zapomíná. Na rozdíl od lokálních proměnných, které bez vlastní inicializace nelze použít (a vývojové prostředí na to důrazně upozorňuje), atributy použít bez zadání hodnoty lze velmi dobře, avšak se s tím musí počítat. Jedná se o funkčnost, se kterou je třeba počítat. Není nezbytně nutné nastavovat specificky atributu hodnotu 0, protože té hodnoty bude automaticky nabývat i tak. Na druhou stranu je pravda, že uvedením této hodnoty můžeme učinit kód lépe čitelným pro sebe i případně pro ostatní. Technicky vzato existují tabulky, které uvádějí, jaké jsou konkrétní výchozí hodnoty pro všechny datové typy, ale celou problematiku lze shrnout do tvrzení, že výchozí hodnou je [47]:

- **0** - pro primitivní datové typy (v případě booleanu reprezentovaná hodnotou false)
- **null** – pro referenční datové typy (třídy, výčtový typ, pole)

Efekt automatické inicializace si mohou studenti sami okamžitě vyzkoušet tak, že vytvoří novou třídu s názvem Žák a deklarují v ní 5 atributů objektu: jméno a příjmení (samostatně s tím, že jim sdělíme, že datovým typem pro text je třída String), rok narození, studijní průměr a zda studuje. Výsledná třída pak může vypadat následovně:

```
class Žák {
    String jméno;
    String příjmení;
    int rokNarození;
    double studijníPrůměr;
    boolean studuje;
}
```

Pomocí zapsané třídy poté vytvoří instanci třídy kliknutím pravým tlačítkem na třídu a vybráním volby „new Žák()“. Jméno odkazu na objekt není z hlediska programu podstatné, tedy postačí ponechat automaticky vygenerované a klepnout na tlačítko OK. Dole na liště objektů poté opět pravým tlačítkem myši a zvolením volby „Prohlížet“ vyvoláme dialogové okno zobrazující aktuální hodnoty atributů daného objektu tak, jako na obrázku níže (Obr. 12).



Obrázek 12 – Prohlížení automaticky inicializovaných atributů objektu v BlueJ

Tady taktéž můžeme kontrolovat, zda si žáci pojmenovali atributy relevantním způsobem. Pochopitelně zde není dobré vyžadovat exaktní názvy proměnných, pokud se nachází v přijatelných mezích

Nejběžnějším způsobem, jak přiřadit proměnné hodnotu, je tzv. „přiřazovací příkaz“, jež má obecný tvar [48]:

proměnná = výraz;

Zde bych použil určitý příměr k matematice, pomocí které si žák tento výraz může lépe představit. Avšak na rozdíl od klasické algebry je nutno zdůraznit, že na levé straně přiřa-

zovacího operátoru (v případě Javy symbolu rovnítka) musí být vždy a pouze proměnná, která určuje místo v paměti, kam se uloží výsledek výpočtu výrazu, zapsaného na straně pravé. Zde by bylo na místě rozvést, co je konkrétně myšleno slovem „výraz“, neboť i to má svá specifika, která musí být jasná. V prvé řadě bych uvedl, že se výraz skládá z kombinace [49]:

- Proměnných
- Literálů
- Operátorů
- Volání metod
- Jednoduchých závorek

Opět lze ukázat několik příkladů ke každé výše zmíněné kategorii, ale podrobnější specifika a úskalí bych uvedl až v pozdějších hodinách nebo při konkrétních situacích.

Pozitivní ovšem je, že vysvětlit postup, s jakým se vyhodnocují výrazy, nebude příliš složité, neboť zde funguje stejný postup jako v matematice s určitými rozšířeními a operace, které z matematiky nejsou úplně zřejmé.

Před závěrečným úkolem hodiny, se musí žáci dozvědět, jakým způsobem v kódu vytvoří novou instanci třídy. Pro mnohé nebude překvapením, že tak učiní stejným příkazem, který vídají při vytváření objektů v prostředí BlueJ a to konkrétně volání příslušného konstruktoru zvolené třídy pomocí operátoru ***new***:

```
Barva červenáBarva = new Barva(255, 0, 0);
```

Tento příkaz již kombinuje již získané znalosti a to konkrétně deklarace proměnné *červenáBarva* datového typu Barva její okamžitá inicializace pomocí přiřazovacího příkazu. Jako výraz, který se vyhodnocuje na pravé straně, zde vidíme volání metody (konkrétně konstruktoru třídy Barva o třech celočíselných parametrech). Výsledkem po vyhodnocení tohoto výrazu je odkaz na sestavenou instanci třídy Barva o zadaných parametrech, který je následně přiřazen do proměnné *červenáBarva*.

Deklarovanou a inicializovanou proměnnou *červenáBarva* nyní využijeme pro ukázku, jak se přistupuje k atributům objektu a jeho metodám:

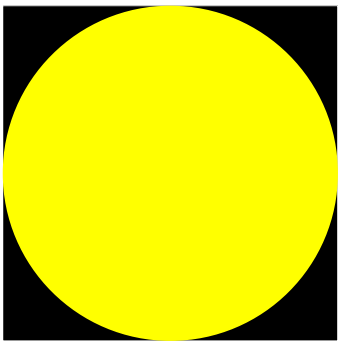
```
červenáBarva.red
```

Tímto výrazem obdržíme hodnotu červené složky barevného modelu RGB, kterou jsme zadali při vytváření instance, tedy 255. Pochopitelně se nejedná o příkaz, který by šel vykonat, ale o výraz, který musíme zakomponovat do jiného příkazu, ať už přiřazovacího nebo např. do volání funkce.

```
červenáBarva.getČervená()
```

Podobně jako u atributu *red*, můžeme pomocí tečkového operátoru zavolat metodu *getČervená()* a obdržet stejnou hodnotu.

Závěrečný úkol hodiny, který budeme několik dalších hodin rozvíjet, spočívá pouze v aplikaci výše uvedených znalostí. Úkolem žáků je vytvořit třídu pojmenovanou Lampa, která bude graficky odpovídat obrázku (Obr. 13).



Obrázek 13 – Grafická reprezentace třídy Lampa

Během hodiny i při kontrole úkolů považuju za důležité hlídat, aby měli studenti kód čitelný a dobře stylizovaný za použití tabulátorů a případného rozdělení na více řádků. Ke kontrole tohoto úkolu dojde na začátku následující hodiny, kde bude podstatné, aby měli všichni studenti úkol splněn, neboť se s vytvořeným kódem bude dále pracovat.

Výsledná třída by měla vypadat přibližně takto:

```
class Lampa {
    Ovál světlo = new Ovál(100, 100, Barva.ŽLUTÁ);
    Obdélník obal = new Obdélník(100, 100, Barva.ČERNÁ);
}
```

6.3 Vlastní třída (Metody)

V návaznosti na minulou hodinu je cílem této hodiny rozšířit povědomí žáků o metodách a konstruktorech a rozšířit vytvořenou třídu Lampa o tyto prvky.

Metoda v Javě představuje způsob, jak definovat řadu operací, kterou daný objekt dokáže vykonat a kterou může daná třída (popř. jiné třídy) volat, aby se operace vykonaly.

```
návratovýTyp názevMetody(deklarace parametrů) {
   // tělo metody
}
```

Základní syntaxe metody se skládá z typu návratové hodnoty, názvu metody a deklarace jejích parametrů. Metoda mít pochopitelně parametry nemusí, což poznáme podle prázdných závorek. Stejně tak nemusí mít návratovou hodnotu, což indikuje použití prázdného datového typu *void*. Pro pojmenovávání platí výše uvedená pravidla pro tvorbu identifikátorů, ale dále je konvencí, že se metody pojmenovávají především slovesy v rozkazovacím způsobu. Má-li metoda deklarovaný typ návratové hodnoty jiný než *void*, musí vždy obsahovat návratový příkaz pomocí klíčového slova *return*. Toto klíčové slovo vždy ukončuje běh metody a může být umístěno i na více místech metody, pokud platí, že lze ke všem těmto návratovým příkazům za určitých okolností dojít. Klíčové slovo *return* můžeme využít i v metodách s návratovou hodnotou *void*, ale jeho použití je volitelné. Návratový příkaz má tvar:

```
return výraz;
```

Výsledek výrazu v tomto příkazu musí mít stejný datový typ, jaký je deklarovaný v hlavičce metody.

Úkol pro žáky: Ve vytvořené třídě Lampa vytvořte bezparametrickou metodu getBarva, která vrací barvu světla lampy. K zjištění barvy můžou využít stejnojmennou metodu Oválu. Vzor výsledné metody následuje:

```
Barva getBarva() {
   return světlo.getBarva();
}
```

6.3.1 Konstruktory

Konstruktor naproti tomu představuje speciální typ metody, která nemá návratovou hodnotu, a proto její typ není třeba deklarovat.

```
NázevTřídy(deklarace parametrů) {
   // tělo konstruktoru
}
```

Konstruktor slouží k inicializaci atributů vytvářeného objektu a zavolat jej můžeme pouze pomocí operátoru *new* odkudkoliv nebo pomocí klíčového slova *this* z konstruktoru té samé třídy (viz přetěžování konstruktorů). Konstruktor může mít stejně jako metoda parametry, které se deklarují stejně, jako u metod.

Přetěžování konstruktorů je další podstatnou funkcí, kterou můžeme použít při vytváření třídy. Princip spočívá ve vytváření dalších konstruktorů s jinými parametry. Cílem je vytvořit konstruktor, který obsahuje inicializaci všech potřebných atributů pomocí parametrů a ostatní konstruktory buď přímo, nebo postupně využívají vytvořených konstruktorů. Smyslem tohoto přístupu je vytvořit kód, kde jsou atributy inicializovány pouze jedním konstruktorem, díky čemuž je snazší provádět změny.

Se znalostí konstruktorů a jejich přetěžování mají žáci za úkol ve třídě Lampa nejprve přesunout inicializaci atributů do bezparametrického konstruktoru a poté vytvořit konstruktor s jedním parametrem, kde bude možné zadat barvu světla Lampy. A konečně využít tento vytvořený konstruktor v bezparametrickém konstruktoru. Řešení těchto úkolů by mohl vypadat takto:

```
Lampa(Barva barvaSvětla) {
  obal = new Obdélník(100, 100, Barva.ČERNÁ);
  světlo = new Ovál(100, 100, barvaSvětla);
}

Lampa() {
  this(Barva.ŽLUTÁ);
}
```

U metod funguje přetěžování na stejném principu jako v případě konstruktorů s tím rozdílem, že metody vzájemně musíme volat jejich celým názvem namísto pouhého klíčového slova *this*.

Přetěžování metod můžeme na naší třídě Lampa demonstrovat implementací dvou metod pro nastavení barvy světla:

```
void setBarva(Barva nováBarva) {
   světlo.setBarva(nováBarva);
}

void setBarva(int červená, int zelená, int modrá) {
     setBarva(new Barva(červená, zelená, modrá));
}
```

Žáci by taktéž měli porozumět signatuře metody, jež slouží jako jednoznačný identifikátor dané metody. Signatura se v Javě skládá z názvu a parametrů metody, resp. jejich počtu, pořadí a datových typů). Typ návratové hodnoty není součástí signatury. To je také důvod proč:

`int getČíslo()` a `double getČíslo()`

jsou z pohledu Javy stejné metody a překladač takovou třídu odmítne přeložit.

6.4 Atributy třídy a objektu

Ve třídě lze kromě atributů objektu, které má každá instance třídy vlastní a jsou přístupné pouze přes samotnou instanci třídy, i atributy třídy. Tyto atributy jsou pro všechny instance dané třídy společné a jejich změna ovlivní všechny instance třídy, které jsou již vytvořené, tak i ty, které budou teprve vytvořeny. Atribut třídy se deklaruje pomocí klíčového slova *static* uvedeném před datovým typem proměnné. [46]

```
static int počet;
```

Použití tohoto druhu atributu bychom ale měli, s výjimkou konstant, omezit na minimum.

Pokud se nemá hodnota atributu měnit, lze toto indikovat použitím klíčového slova *final* a to podobně jako v případě atributu třídy před datovým typem.

```
final int počet;
```

Označit takto atribut má řadu výhod. Zabrání se tak úpravě atributu, ať již náhodou nebo úmyslně. Za použití tohoto modifikátoru lze v případě primitivních datových typů odpustit, že je ponechán v rozporu s principem zapouzdření přímý přístup k atributu.

Velmi užitečné je oba tyto modifikátory zkombinovat a vytvořit tzv. „konstantu“. Tu pojmenováváme, na rozdíl od jiných atributů, velkými písmeny a jednotlivá slova oddělujeme podtržítkem.

```
static final int VÝCHOZÍ_POČET;
```

S využitím těchto znalostí, dostanou žáci za úkol vytvořit konstantu pro výchozí barvu Lampy a tuto konstantu zakomponovat do bezparametrického konstruktoru. Výsledný část definice třídy by měla vypadat poté následovně:

```
static final Barva VÝCHOZÍ_BARVA_LAMPY = Barva.ŽLUTÁ;

Lampa() {
   this(VÝCHOZÍ_BARVA_LAMPY);
}
```

6.5 Primitivní datové typy

Lehce rozvést téma primitivních datových typů, ačkoliv je studenti v menší míře již používali, bychom měli taktéž probrat lépe dříve nežli později. Primitivní datové typy jsou nositeli surových dat a dělí se na 4 typy celých čísel (dle rozsahu), 2 typy desetinných čísel (taktéž podle přesnosti), jeden logický typ a jeden typ pro znak.

Pro celá čísla můžeme použít jeden ze čtyř datových typů (long, int, short a byte) o rozmezí hodnot, které v nich můžeme uložit, uvedených v tabulce (Tab. 7).

Datový typ	Rozsah	Minimální hodnota	Maximální hodnota
long	64 bitů	-2^{63}	$2^{63}-1$
int	32 bitů	$-2\,147\,483\,648$ (-2^{31})	$2\,147\,483\,647$ $(2^{31}-1)$
short	16 bitů	$-32\,768$ (-2^{15})	$32\,767$ $(2^{15}-1)$
byte	8 bitů	-128	127

Tabulka 7 – Rozsahy celočíselných primitivních datových typů [47]

Nejběžnějším typem čísla je int, což je také datový typ čísla, na které je převeden běžně zapsaného literálu čísla (např. 100), a jeho rozsah stačí pro všechny běžné potřeby. Pokud je rozsah int nedostatečný, musíme pro velká čísla použít datový typ long, pro jehož zápis musíme v literálu uvést na konci „L“ (tedy 100L). Máme-li pro číselná data v zařízení vel-

ké omezení na paměť, budeme uvažovat o použití datového typu short, do něhož přiřazujeme čísla stejným literálem, jako v případě int-u, je-li splněn rozsah:

```
short číslo = 100; // v pořádku – 100 je short
short číslo2 = 300; // chyba, 300 je mimo rozsah, je to int
```

Datový typ byte pak používáme pouze při operacích se surovými daty v souboru a pro jeho ruční zápis platí stejné pravidlo jako u short-u.

U celých čísel můžeme provádět běžné aritmetické operace, na které jsme zvyklí ze základní matematiky, jako jsou sčítání, odčítání, násobení, dělení a lehce netradiční operace modulo (vrácení zbytku po celočíselném dělení). U dělení si ale musíme dát pozor na fakt, že výsledkem dělení dvou celých čísel je v aritmetice Javy opět celé číslo a tím pádem:

```
int výsledek = 5 / 2;
```

přiřadí do proměnné „výsledek“ hodnotu 2, neboť desetinná část z očekávaného výsledku 2,5 je zahozena. Chceme-li tedy obdržet správný výsledek, nejen, že musíme jako datový typ výsledku deklarovat desetinné číslo (např. double), ale musíme navíc, což je podstatnější, změnit datový typ jednoho z čísel na desetinné. Nejjednodušší způsob, jak toho dosáhnout u konkrétního čísla, je zapsat jej jako desetinné:

```
double výsledek = 5.0 / 2;
```

Výsledkem dělení dvou čísel, z nichž jedno je desetinné, je vždy desetinné číslo rozsahu, jaký má číslo s největším rozsahem.

Je třeba si také uvědomit, co se stane, když manipulujeme s číslem za hranicemi jeho rozsahu. Máme-li celé číslo typu int s přiřazenou maximální hodnotou jeho rozsahu (2 147 483 674) a přičteme-li k takovému číslu 1, dojde k tzv. přetečení (angl.. overflow) a výsledná hodnota bude mít hodnotu -2 147 483 648.

Při manipulaci čísli, také dochází k automatické konverzi. Máme řadu datových typů:

byte → *short* → *int* → *long* → *float* → *double*

Je-li cílovým datovým typem typ napravo od zdrojového datového typu, je provedena automatická konverze bez ztráty přesnosti. Např. při použití proměnných „a“ a „b“:

```
int a = 5;
double b = a;
```

je při přiřazování hodnoty „a“ do „b“ automaticky „a“ převedeno na double.

Konverze opačným směrem je taktéž možná, ale musíme ji provést manuálně pomocí přetypování a není zaručena přesnost. Tedy použijeme-li podobný příklad:

```
double b = 5.5;
int a = (int)b;
```

U převodu 5.5 z „b“ do „a“ tímto došlo ke ztrátě přesnosti a převedení pouze hodnoty 5.

Pokud máme zájem uchovávat desetinná čísla pro přesnost, používáme běžně datový typ *double*, což je i datový typ běžně používaného zápisu desetinného čísla. Máme-li ale omezenou paměť a uchováváme velké množství desetinných čísel, můžeme se při určité ztrátě přesnosti uchýlit k použití datového typu *float*.

6.6 Podmínky

Rozhodovací blok podmínky je základní způsob, jak umožnit objektům měnit své chování na základě zvoleného aspektu. Syntaxi podmínky můžeme použít buďto v základní podobě:

```
if(logický výraz) příkaz;
```

alternativně i místo jednoho příkazů, lze použít blok se sadou příkazů:

```
if(logický výraz) {příkaz1;příkaz2;}
```

Logický výraz je takový výraz, jehož výsledkem je hodnota typu boolean, tedy true (pravda) nebo false (nepravda). Je-li hodnota vyhodnoceného výrazu true, vykoná se příkaz nebo sada příkazů uvedených za podmínkou. Pokud je nutné provést zároveň sadu příkazů, které se vykonají, nesplní-li se podmínka, použijeme rozšířený zápis podmínky:

```
if(logický výraz) {příkaz1;příkaz2;} else {příkaz3;příkaz4;}
```

Pochopitelně za jeden nebo za oba bloky příkazů ve složených závorkách lze dosadit pouze jediný příkaz bez složených závorek, ale vždy zakončený středníkem.

6.6.1 Srovnávací operace

Abychom ale měli co v podmínkách vyhodnocovat, musíme znát alespoň základní srovnávací operace a operátory. Pro účely porovnání číselných primitivních datových typů máme na výběr z těch běžných operátorů, jakými jsou: > (větší než), < (menší než), >= (větší nebo rovno), <= (menší nebo rovno), == (jsou rovny) a != (jsou různé). Výsledkem takové operace jsou logická hodnota true nebo false, např.:

```
5 > 3          a == 10          a != b
```

Pracujeme-li s objekty, můžeme je taktéž porovnávat, resp. můžeme pomocí operátorů == a != porovnat, zda jsou porovnávané odkazy totožné. Tuto odlišnost je třeba si uvědomit, neboť pro porovnání obsahu samotných objektů musíme použít metodu equals, kterou každá třída má. Pro ilustraci:

```
new Barva(0,0,0) == new Barva(0,0,0)
```

bude vyhodnoceno jako false (nepravda), protože jsou vytvořeny (ač se stejnými parametry) 2 různé objekty. Pro porovnání faktické shody mezi těmi to objekty použijeme tedy metodu equals:

```
new Barva(0,0,0).equals(new Barva(0,0,0))
```

Tento zápis již vrátí výsledek true (pravda), obě barvy jsou černé.

6.6.2 Logické operace

Abychom nemuseli složitě opakovat konstrukce podmínek, pokud bychom si přáli zkontrolování splnění více podmínek, můžeme využít logických operací. První běžnou logickou operací je logický součin, v běžné mluvě vyjádřen spojkou „a“. Pro tuto logickou operaci se používá operátor && a operace má 2 operandy (2 vstupy). Výsledkem této operace je true (pravda) pouze pokud jsou oba vstupy vyhodnoceny jako true. V opačném případě se operace vyhodnotí jako false. Např.: zjišťujeme, je-li proměnná x větší než 5 a zároveň menší než 10, použijeme následující logický výraz:

```
x > 5 && x < 10
```

Další logickou operací je logický součet, běžně vyjádřen spojkou „nebo“. Tuto logickou operaci indikuje operátor ||. Výsledná hodnota takové operace je false pouze pokud oba vstupy mají hodnotu false. V opačném případě vrací operace hodnotu true. Chceme-li tedy, podobně jako u předchozího příkladu, zjistit, zda je proměnná x menší než 5 nebo větší než 10, bude logický výraz vypadat takto:

```
x < 5 || x > 10
```

Poslední zásadní logickou operací je logická negace indikovaná operátorem vykřičníku (!). Tato operace jednoduše převrátí logickou hodnotu daného výrazu na opačnou. Např.: Pro negaci proměnné „x“ zapíšeme výraz jednoduše !x, což se v podmínce projeví takto:

```
if(!x) příkaz;
```

Blok příkazu vykonávaný při splnění podmínky může pochopitelně obsahovat i deklarace a definice dalších proměnných. Tyto proměnné jsou ale platné pouze v rámci bloku, ve kterém byly deklarovány:

```
void uprav(int x) {
   if(x > 5) {
      int y = 2;
   }
}
```

Po uzavření bloku za proměnnou „y“ pomocí složené závorky, přestává „y“ v paměti existovat a nelze jej dále používat. Abychom jej mohli používat mimo podmínku, museli bychom jej deklarovat mimo podmínku a v případě tohoto příkladu ještě navíc inicializovat, protože kdyby se nesplnila podmínka, tak by proměnná „y“ neměla hodnotu.

```
void uprav(int x) {
   int y = 10;

   if(x > 5) {
      y = 2;
   }
}
```

Kromě tohoto omezení platnosti také dvě proměnné uvnitř metody nemůžou mít stejný identifikátor. Zatímco k atributům objektu můžeme přistoupit pomocí klíčového slova this a k atributům třídy zase přes název třídy, není možné ve vnořených blocích metody při použití stejně pojmenovaných proměnných rozlišit, kterou proměnnou použít.

6.7 Výčtový datový typ a String

Při návrhu a implementaci aplikace běžně potřebujeme výčet konkrétního typu hodnot. Do doby než Java obsahovala výčty, se k tomuto účelu musely využívat třídní konstanty. Takový zápis byl zdlouhavý a často vyžadoval vytváření pomocných tříd, které plnily roli nositelů informace. Po zavedení výčtu bylo možné zápis zkrátit o mnohé modifikátory, protože se jejich role předpokládala. Vezměme si srovnání následující třídy Den v porovnání s výčtovým typem Den.

```
class Den {
   static final Den PONDĚLÍ = new Den("Po");
   static final Den ÚTERÝ = new Den("Út");
   static final Den STŘEDA = new Den("St");
   static final Den ČTVRTEK = new Den("Čt");
   static final Den PÁTEK = new Den("Pá");
   static final Den SOBOTA = new Den("So");
   static final Den NEDĚLE = new Den("Ne");

   String zkratka;

   private Den(String zkr) {
      zkratka = zkr;
   }
}
```

```
enum Den {
   PONDĚLÍ("Po"),
   ÚTERÝ("Út"),
   STŘEDA("St"),
   ČTVRTEK("Čt"),
   PÁTEK("Pá"),
   SOBOTA("So"),
   NEDĚLE("Ne");

   String zkratka;

   Den(String zkr) {
      zkratka = zkr;
   }
}
```

Na příkladu je vidět k jak masivnímu zkrácení kódu došlo, ačkoliv oba plní tutéž roli. Stejně jako běžné třídy, můžou i výčty obsahovat metody a to třídní i instanční.

Výhoda výčtu spočívá ve zjednodušení do konkrétního konceptu, kde se navíc dá automaticky postupně přistupovat k prvkům výčtu, jakoby byly v seřazeném seznamu, tedy přecházet od jednoho k druhému.

Již v minulosti jsme se zmiňovali o tom, že pro ukládání znaku textu se používá primitivní datový typ char. Jeden znak ale běžně nestačí a pro tento účel slouží třída String, která není jen obalová třída, ale specifikace Javy s ní počítá. Usnadňuje nám totiž zápis při jejím vytváření a díky tomu stačí zapsat pouze textový literál s uvozovkami, aniž bychom museli vypisovat jednotlivé znaky po jednom:

```
String text = "ahoj";
```

Aby nebyla manipulace z textem tak komplikovaná, můžeme jednotlivé textové řetězce spojovat (zřetězit) pouze za pomocí operátoru +, čímž ušetříme čas pro psaní složitějších konstrukcí.

```
String a = "ahoj";
String s = "světe";
String v = a + " " + s;
```

Díky tomu, že je String třída, nabízí řadu metod, pro manipulaci s daným textem, jako zjištění délky (počtu znaků), vrácení podmnožiny z textu nebo nalezení umístění hledaného úryvku textu.

6.8 Balíčky a přístupové modifikátory

Třídy se v Javě organizují do tzv. „balíčků“, což je zjednodušeně složková hierarchie. Abychom indikovali, že je třída součástí určitého balíčku, nestačí, aby byla pouze ve správné složce, musí se tato skutečnost indikovat ještě před deklarací třídy v souboru dané třídy:

```
package mojetridy.kresleni;

class Obdélník {}
```

Podsložky se zde namísto klasického nebo obráceného lomítka značí tečkou.

Protože nechceme vždy umožnit přístup k našim atributům a metodám, musíme mít možnost tento přístup omezit. K tomu slouží přístupové modifikátory, z nichž nejběžnějšími jsou značeny klíčovými slovy public a private.

Public označuje plný přístup odkudkoliv a můžeme jím označit atributy, metody nebo třídy. Označujeme takto především takové metody, které mají být využívány jinými třídami v okolním kódu.

```
public int počet = 5;
public int getPočet() {
     return počet;
}
```

Private naproti tomu označuje minimální přístup, tedy přístup pouze ze třídy samotné. Měli bychom takto v zájmu principu zapouzdření označovat především veškeré modifikovatelné atributy. Zároveň bychom tak měli označit i pomocné metody, které nejsou mimo danou třídu využitelné.

```
private int počet = 5;
private void přepočítej() {
     počet = 10;
}
```

V předešlých příkladech bylo možné si všimnout, že jsme u metod ani atributů žádné přístupové modifikátory neuváděli. To ale neznamená, že žádné nemají, naopak. Není-li přístupový modifikátor uveden, znamená to, že je použit výchozí přístupový modifikátor (tzv. „package-private“). Ten říká, že je daný prvek přístupný pouze z třídy nebo balíčku, ve kterém se třída nachází, ale nikoliv mimo něj.

6.9 Rozhraní (interface)

Rozhraní jsou velmi důležitou součástí designu aplikace. Umožňují navrhnout obecně funkčnost, která dostane konkrétní podobu až později a tím zrychlit postup při návrhu i vývoji. Konkrétně předepisují hlavičky metod, které jsou abstraktní, tedy neobsahují konkrétní podobu příkazů, které se budou při její volání provádět.

Příklad rozhraní může být řekněme Zbožíznalec. V tomto rozhraní bude deklarováno, že umí předat zboží podle názvu. Třída Prodavač pak rozhraní Zbožíznalec tzv. „implementuje“, čímž musí pro předepsanou schopnost definovat postup, jakým toho dosáhnout. Díky tomu, když budeme jednat s někým, kdo implementuje rozhraní Zbožíznalec, víme jaký schopnosti od něj očekávat, aniž by nás zajímalo, o koho se vlastně jedná (o jakou třídu). Prakticky by vypadal zápis takto:

```
interface Zbožíznalec {
      Zboží přinesZboží(String název);
}

class Prodavač implements Zbožíznalec {

      Zboží přinesZboží(String název) {
            // zamyslet se zda zboží máme
            // uvědomit si kde zboží je
            // přinést a předat zboží
      }
}
```

Zde si můžeme všimnout, že v rozhraní končí deklarace metody přinesZboží středníkem místo těla metody. Toto je pro rozhraní typické, neboť je taková metoda tzv. „abstraktní“. Z praktického hlediska tedy z rozhraní nelze vytvořit přímou instanci, ale musí být realizováno vždy skrze běžnou třídu.

Výhodou rozhraní je také to, že třída může implementovat více rozhraní na rozdíl od klasické dědičnosti.

Pro procvičení vytváření rozhraní dostanou žáci za úkol vytvořit rozhraní ZdrojSvětla a předepsat v něm 2 metody bez návratové hodnoty:

- rozsviťSe() – metoda změní barvu světla na původní barvu
- zhasniSe() – metoda změní barvu světla na barvu obalu

Při implementaci metod narazíme na malý problém, odkud zjistit původní barvu světla. Pro tento účel navrhuju řešení si barvu při vytváření objektu uložit do atributu. Pochopitelně nejprve vhodné řešení zkusíme hledat studenty. Rozhraní a podstatné části třídy by po těchto změnách měly vypadat přibližně takto:

```
interface ZdrojSvětla {
    void rozsviťSe();
    void zhasniSe();
}

class Lampa {
    Ovál světlo;
    Obdélník obal;
    Barva barvaLampy;

    Lampa(Barva barvaLampy) {
        obal = new Obdélník(100, 100, Barva.ČERNÁ);
        světlo = new Ovál(100, 100, barvaLampy);
        this.barvaLampy = barvaLampy;
    }

    void rozsviťSe() {
       světlo.setBarva(barvaLampy);
    }

    void zhasniSe() {
       světlo.setBarva(obal.getBarva());
    }
}
```

6.10 Dědičnost

Dalším z podstatných konceptů OOP je dědičnost. Ta umožňuje přebírat nesoukromé atributy a metody z jiné třídy do vlastní třídy. Tyto atributy pak může potomek ponechat, měnit jejich hodnotu nebo zděděné metody volat a atributy používat.

Všechny třídy v Javě podléhají dědičnosti, protože každá třída je, ať už přímo nebo nepřímo potomkem třídy java.lang.Object, avšak přímého rodiče může mít každá třída pouze jednoho. Příklad jednoduché syntaxe deklarace páru rodič-potomek vypadá takto:

```
class Rodič {
      void promluv(String slovo) {
         řekni(slovo);
     }
…

class Potomek extends Rodič {
      // Potomek nyní umí taktéž
      // promluvit
…
```

Jak již bylo řečeno, potomek může implementaci zděděné metody upravit a to např.:

```
class Rodič {
    void promluv(String slovo) {
        řekni(slovo);
    }
…

class Potomek extends Rodič {
    @Override // pouze poukazuje na přepsaní zděděné metody
    void promluv(String slovo) {
        zakřič(slovo);
    }
…
```

Pokud se zavolá v potomkovi metoda promluv, zavolá se, namísto rodičovy verze, metoda zakřič. Chceme-li ale sami zavolat rodičovskou metodu, musíme k tomu použít klíčové slovo super:

```
class Rodič {
    void promluvit(String slovo) {
        řekni(slovo);
    }
…

class Potomek extends Rodič {

    @Override // pouze poukazuje na přepsaní zděděné metody
    void promluvit(String slovo) {
        super.promluv(slovo + "!!");
    }
…
```

Díky dědičnosti vzniká mezi třídami jasná hierarchie, která je jedním z aspektů dalšího principu OOP, kterým je polymorfismus.

6.10.1 Polymorfismus

Tento koncept vychází ze struktury dědičnosti a vlastností rozhraní, díky kterým je možné vnímat objekty, jako objekty různých typů. Rozšiřuje-li tedy třída Potomek třídu Rodič, můžeme s Potomkem nakládat jako s rodičem, protože bude obsahovat všechny veřejné metody a atributy, jaké měl rodič:

```
Rodič rodič = new Rodič();
rodič.řekni("ahoj");

Rodič potomek = new Potomek();
potomek.řekni("taky zdravím");
```

V obou případech vím, že můžeme zavolat metodu „řekni“ s parametrem typu String a ačkoliv se metody nemusí vykonat stejně, volání nic nebrání.

Stejný princip funguje i v případě rozhraní, kdy víme-li, že třída implementuje určité rozhraní, můžeme ji typově považovat za dané rozhraní a vyvolávat z ní předepsané metody.

ZÁVĚR

Hlavním cílem mé práce bylo sestavit didaktické pomůcky ve formě prezentačních materiálů s důrazem na obsažení základních znalostí a konceptů programování pomocí platformy Java.

V teoretické části bylo cílem zdůraznit výhody vyučování programování pomocí jazyka Java a jeho vývojové platformy. Dále pak zhodnotit a vybrat vhodné vývojové prostředí, které studentům umožní nejsnazší pochopení probírané látky. Nejvhodnějším prostředím byl shledán BlueJ, vývojové prostředí, jež je od počátku vyvíjeno pro účely edukace. Jeho volná dostupnost a řada nástrojů, které obsahuje, a jsou pro výuku vhodné, z něj činil nejlepšího kandidáta.

V praktické části je obsažen popis uživatelského prostředí aplikace BlueJ. A dále pak popis a vysvětlení jednotlivých zvolených základních témat, které studenta uvádějí do problematiky objektově orientovaného programování a jazyka Java.

SEZNAM POUŽITÉ LITERATURY

[1] Rámcový vzdělávací program pro obor vzdělání 18-20-M/01 Informační technologie. *Národní ústav odborného vzdělávání* [online]. 2008 [cit. 2008]. Dostupné z: http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf

[2] BV, TIOBE. TIOBE Index for March 2018. *TIOBE - software quality company* [online]. 2018 [cit. 2018]. Dostupné z: https://www.tiobe.com/tiobe-index/

[3] STACKOVERFLOW.COM, . Developer Survey Results 2018. *Stackoverflow.com* [online]. 2018 [cit. 2018]. Dostupné z: https://insights.stackoverflow.com/survey/2018/

[4] SURVS.COM, . Git User's Survey 2016 (except ANON). *Survs.com* [online]. 2016 [cit. 2018]. Dostupné z: https://survs.com/report/nz2odu1spl

[5] SCHILDT, Herber. *Java 7 - Výukový kurz*. 1. vydání. Brno: Computer Press, 2012.

[6] MULLIS, Alex. How to install the Android SDK (Software Development Kit). *Android Authority* [online]. 2017 [cit. 2018]. Dostupné z: https://www.androidauthority.com/how-to-install-android-sdk-software-development-kit-21137/

[7] IDC: Smartphone OS Market Share. *IDC: The premier global market intelligence firm* [online]. b.r. [cit. 2017]. Dostupné z: https://www.idc.com/promo/smartphone-market-share/os

[8] SHALAJEV, Oleg. Top Java Web Frameworks Popularity Index. *RebelLabs Archive* [online]. 2017 [cit. 2017]. Dostupné z: https://zeroturnaround.com/rebellabs/java-web-frameworks-index-by-rebellabs/

[9] About BlueJ. *BlueJ* [online]. b.r. [cit. 2018]. Dostupné z: https://www.bluej.org/about.html

[10] BlueJ. *BlueJ* [online]. b.r. [cit. 2018]. Dostupné z: https://bluej.org/

[11] ECLIPSE FOUNDATION, Inc. *Open Innovation Community - Eclipse IDE* [online]. b.r. [cit. 2018]. Dostupné z: www.eclipse.org

[12] FOWLER, Martin. Crossing Refactoring's Rubicon. *Martin Fowler* [online]. 2001 [cit. 2018]. Dostupné z: https://martinfowler.com/articles/refactoringRubicon.html

[13] JETBRAINS, . *IntelliJ IDEA: The Java IDE for Professional Developers*. 2018. Dostupné také z: www.jetbrains.com/idea/

[14] A Brief History of NetBeans. *NetBeans IDE* [online]. b.r. [cit. 2018]. Dostupné z: https://netbeans.org/about/history.html

[15] *NetBeans IDE* [online]. b.r. [cit. 2018]. Dostupné z: netbeans.org

[16] BYOUS, Jon. Java Technology: The Early Years. *Sun Developer Network* [online]. 2003 [cit. 2018]. Dostupné z: http://java.sun.com/features/1998/05/birthday.html

[17] Sun Ships JDK 1.1 -- JavaBeans included. *Sun Microsystems* [online]. 1997 [cit. 2018]. Dostupné z: http://www.sun.com/smi/Press/sunflash/1997-02/sunflash.970219.0001.xml

[18] Sun delivers next version of the Java platform. *Sun Microsystems* [online]. 1998 [cit. 2018]. Dostupné z: http://www.sun.com/smi/Press/sunflash/1998-12/sunflash.981208.9.xml

[19] Sun Microsystems releases fastest client-side Java platform to date. *Sun Microsystems* [online]. 2000 [cit. 2018]. Dostupné z: http://www.sun.com/smi/Press/sunflash/2000-05/sunflash.20000508.3.xml

[20] Sun announces latest version of Java 2 Platform Standard Edition. *Sun Microsystems* [online]. 2002 [cit. 2018]. Dostupné z: http://www.sun.com/smi/Press/sunflash/2002-02/sunflash.20020206.5.xml

[21] Version 1.5.0 or 5.0?. *Oracle Help Center* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/1.5.0/docs/relnotes/version-5.0.html

[22] Sun Announces Revolutionary Version of Java Technology – Java Platform Standard Edition 6. *Sun Microsystems* [online]. 2006 [cit. 2018]. Dostupné z: http://www.sun.com/smi/Press/sunflash/2006-12/sunflash.20061211.1.xml

[23] Oracle faces in-depth EU probe over Sun purchase. *Business Standard* [online]. 2009 [cit. 2018]. Dostupné z: http://www.business-standard.com/article/companies/oracle-

faces-in-depth-eu-probe-over-sun-purchase-109090400042_1.html

[24] TECH, Pure. *Java 7* [online]. b.r. [cit. 2018]. Dostupné z: https://puredanger.github.io/tech.puredanger.com/java7

[25] JDK8. *Oen JDK* [online]. 2014 [cit. 2018]. Dostupné z: http://openjdk.java.net/projects/jdk8/

[26] Jshell: The Java Shell (Read-Eval-Print-Loop). *Java Bug System* [online]. 2017 [cit. 2018]. Dostupné z: https://bugs.openjdk.java.net/browse/JDK-8043364

[27] GOETZ, Brian. JEP 286: Local-Variable Type Inference. *OpenJDK*. 2018. Dostupné také z: http://openjdk.java.net/jeps/286

[28] ORACLE, . *Differences between Java EE and Java SE* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

[29] Java EE at a Glance. *Oracle* [online]. b.r. [cit. 2018]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[30] Java® Platform, Standard Edition & Java Development Kit. *Oracle* [online]. 2018 [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/10/docs/api/overview-summary.html

[31] The Java Tutorials. *Oracle* [online]. 2018 [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/tutorial/

[32] NECKÁŘ, Jan. *Algoritmy.net* [online]. 2018 [cit. 2018]. Dostupné z: https://www.algoritmy.net/article/21340/Uvod-1

[33] Java - Největší česká online učebnice. *Itnetwork.cz* [online]. 2018 [cit. 2018]. Dostupné z: https://www.itnetwork.cz/java/

[34] PECINOVSKÝ, Rudolf. Výuka OOP. *Pecinovsky.cz* [online]. 2018 [cit. 2018]. Dostupné z: http://vyuka.pecinovsky.cz/

[35] BLOCH, Joshua. *Effective Java*. 3rd edition. Boston, MA: Addison-Wesley, 2018, s. 416.

[36] URMA, Raoul-Gabriel, Mario FUSCO a Alan MYCROFT. *Java 8 in Action:*

Lambdas, streams, and functional-style programming. Shelter Island: Manning, 2014, s. 424.

[37] SCHILDT, Herbert. *Java The Complete Reference*. 10th edition. New York: McGraw-Hill Education, 2017, s. 1344.

[38] PAVLÍČKOVÁ, Jarmila a Luboš PAVLÍČEK. *Úvod do Javy*. Praha: Oeconomica, 2005, s. 227.

[39] TÖPFER, Pavel. *Algoritmy a programovací techniky*. 2. vydání. Praha: Prometheus, 2010, s. 300.

[40] HEROUT, Pavel. *Java - grafické uživatelské prostředí a čeština*. 2. vydání. Českké Budějovice: KOPP, 2007, s. 381.

[41] PECINOVSKÝ, Rudolf. *Java 8: Úvod do objektové architektury pro mírně pokročilé*. Praha: Grada, 2014, s. 656.

[42] KÖLLING, Michael. The Problem of Teaching Object-Oriented Programming, Part 1: Languages. *Semantic Scholar* [online]. 1999 [cit. 2018]. Dostupné z: https://pdfs.semanticscholar.org/8b84/a8495bd43dc92c5986599c36878db909ba9e.pdf

[43] ORACLE, . *What Is a Class* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/concepts/class.html

[44] STACKIFY.COM, . What are OOP Concepts in Java. *Stackify.com* [online]. 2017 [cit. 2018]. Dostupné z: https://stackify.com/oops-concepts-in-java/

[45] ORACLE, . *Declaring Classes* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/javaOO/classdecl.html

[46] ORACLE, . *Declaring Member Variables* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html

[47] ORACLE, . *Primitive Data Types* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

[48] ORACLE, . *Initializing Fields* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html

[49] ORACLE, . *Chapter 15. Expressions* [online]. b.r. [cit. 2018]. Dostupné z: https://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
AWT	Abstract Window Toolkit
GNU GPL	GNU General Public License
HTML5	HyperText Markup Language 5
IDE	Integrated Development Environment
IPv6	Internet Protocol version 6
JDK	Java Development Kit
JIT	Just-in-time
JRE	Java Runtime Environment
JVM	Java Virtual Machine
OOP	Objektově orientované programování
PHP	PHP Hypertext Preprocessor
REPL	Read-Eval-Print-Loop
RVP	Rámcový vzdělávací program

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: DISK CD OBSAHUJÍCÍ PREZENTACE A PROJEKTY