ČESKÁ ZEMĚDĚLSKÁ UNIVERZITAV PRAZE

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Jazyk Java a nástroj Eclipse

Autor: Bc. Jan Kuník

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Kuník Jan

Informatika

Název práce

Jazyk Java a nástroj Eclipse.

Anglický název

The Java Language and the Eclipse Toolset.

Cíle práce

Cílem této diplomové práce je popis modelovacího nástroje Eclipse prostřednictvím programovacího jazyka Java. Funkce modelovacího nástroje Eclipse budou představeny formou aplikací, které budou v tomto nástroji vytvořeny. Teoretická část bude zaměřena na hlavní aspekty programovacího jazyka Java, které budou poté v praktické části realizovány formou aplikací v Eclipse. Dále bude uveden popis modelovacího nástroje Eclipse a jeho funkcí. Po vytvoření aplikací bude následovat testování jejich funkčnosti. V závěru budou zhodnoceny možnosti prostředí Eclipse pro programování v jazyce Java.

Metodika

objektové programování, využití kompomentového přístupu, zásady analýzy a návrhu softwarových aplikací podel zásad softwarového inženýrství

Harmonogram zpracování

04.2013 - 10.2013 sběr podkladů
11.2013 - 01.2014 tvorba aplikací a testování
02.2014 - 03.2014 finalizace práce

Rozsah textové části

40 - 80 stran

Klíčová slova

java, Eclipse, object-oriented programming, toolset

Doporučené zdroje informací

Pavlíčková J., Pavlíček L.: Vývoj klient/server aplikací v Javě. Praha, Oeconomica 2004. ISBN 8024507919.

Zdeněk Troníček: Učebnice jazyka Java. Praha, Česká technika 2007

Herout Pavel: Učebnice jazyka Java. České Budějovice 2001. ISBN 8072321153

http://fedora.cz/eclipse-integrovane-vyvojove-prostredi-pro-javu-i-dalsi-programovaci-jazyky/

Vedoucí práce

Merunka Vojtěch, doc. Ing., Ph.D.

Termín odevzdání

březen 2014

Elektronicky schváleno dne 30.10.2013

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5.12.2013

Ing. Martin Pelikán, Ph.D.

Děkan fakulty

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Jazyk Java a nástroj Eclipse“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou v práci citovány a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne …….

Poděkování

Rád bych poděkoval doc. Ing. Vojtěchovi Merunkovi, Ph.D. za odborné vedení, rady a čas, který mi během přípravy diplomové práce věnoval. Dále bych rád poděkoval celé své rodině za podporu během studia.

Jazyk Java a nástroj Eclipse

The Java language and the Eclipse toolset

Souhrn

Diplomová práce na téma Jazyk Java a nástroj Eclipse obsahuje úvod do teorie programovacího jazyka Java, který je jedním z nejpoužívanějších programovacích jazyků současnosti. Následuje část, která se věnuje teorii XML a DOM v návaznosti na Javu. Po této části je uvedena stručná historie nástroje Eclipse a základní informace o tomto nástroji. V praktické části je popis prostředí Eclipse a jako příklady jsou v průběhu práce tvořené aplikace, na kterých je prakticky ukázána popsaná teorie.

Summary

This thesis is on the theme Java and Eclipse tool includes an introduction to the theory of the Java programming language, which is one of the most used programming languages today. This chapter is followed by a section that focuses on the theory and XML DOM in relation to Java. Following this section is a brief history of Eclipse tools and basic information about this tool. The practical part is a description of the Eclipse environment, and examples are formed in the course of work applications on which is shown practically described the theory.

Klíčová slova: Java, Eclipse, objektově orientované programování, toolset

Keywords: Java, Eclipse, object-oriented programming, toolset

Obsah

Seznam obrázků

Seznam Tabulek

1 Úvod

Jazyk Java patří mezi skupinu programovacích jazyků, které vycházejí z programovacího jazyka C. Java je objektově orientovaný programovací jazyk, který byl vyvinut firmou Sun Microsystems. Tato firma programovací jazyk představila 23. května 1995. Java je jedním z nejpoužívanějších programovacích jazyků na světě především díky svojí přenositelnosti, neboli nezávislosti na platformě, na které bude program spouštěn. V současné době je Java dostupná jako open source.

V první části práce jsou popsány teoretická východiska týkající se nástroje Eclipse a programovacího jazyku Java, včetně úvodu do teorie Javy. V této části jsou popsány základní prvky využité ve vlastní části práce.

Ve vlastním řešení je nejprve popsána instalace nástroje Eclipse a jeho první spuštění doplněné obrazovým materiálem pro jednodušší orientaci. Dále je popsáno pracovní prostředí nástroje Eclipse, po kterém následuje popis založení nového projektu. Na tuto část navazuje ukázka tvorby aplikace ve formě diáře s ukládáním poznámek do souboru typu .xml.

Nástroj Eclipse je vývojová platforma a stejně jako jazyk Java, je open source. Eclipse je díky flexibilnímu návrhu platformy možné za pomoci pluginů rozšířit o možnosti programování například v jazycích C++ nebo PHP.

Eclipse v základní verzi obsahuje integrované prostředky pro standardní práci s programovacím jazykem Java. Jedná se například o debugger a kompilátor. Ostatní potřebné doplňky jsou dostupné pomocí zmíněných pluginů.

Pro vypracování teoretických východisek bylo využíváno aktuálních dostupných tuzemských a zahraničních knižních zdrojů doplněných o zdroje elektronické.

2 Cíl práce a metodika

Cíl práce

Cílem této diplomové práce je popis modelovacího nástroje Eclipse prostřednictvím programovacího jazyka Java. Funkce modelovacího nástroje Eclipse budou představeny formou aplikací, které budou v tomto nástroji vytvořeny.

Teoretická část bude zaměřena na hlavní aspekty programovacího jazyka Java, které budou poté v praktické části realizovány formou aplikací v Eclipse. Dále bude uveden popis modelovacího nástroje Eclipse a jeho funkcí.

Metodika

Metodika práce bude zaměřena na objektové programování, využití komponentového přístupu, zásady analýzy a návrhu softwarových aplikací podle zásad softwarového inženýrství.

Po vytvoření aplikace bude následovat testování jejich funkčnosti. Toto ověření funkčnosti bude provedeno pomocí nástroje debugger, integrovaného v Eclipse. V závěru budou zhodnoceny možnosti prostředí Eclipse pro programování v jazyce Java.

V první kapitole (3.1) literární rešerše jsou popsána jednotlivé využívané programovací paradigmata.

Ve druhé kapitole (3.2) jsou uvedeny základní informace o programovacím jazyku Java, který je použit v praktické části práce pro tvorbu aplikací.

V kapitole nazvané XML (3.3) jsou popsána teoretická východiska o značkovacím jazyku a standardu doporučovanému organizací W3C k ukládání dat, po které následuje kapitola o DOM (3.4), neboli Document Object Model což je aplikační rozhraní.

V kapitole Eclipse (3.6) je stručně popsána historie modelovacího nástroje Eclipse a základní informace o tomto nástroji.

Po těchto kapitolách následuje vlastní práce, ve které je popsána instalace (4.1) a první spuštění nástroje Eclipse (4.2), po kterém následuje popis pracovního prostředí modelovacího nástroje Eclipse (4.3). Dále následují kapitoly popisující import (4.4) a

export (4.5) souborů, po kterých následuje popis postupu založení nového projektu, který je nutný pro vytváření aplikací.

V kapitole Vytvoření aplikace s databází (4.7) je vytvořena aplikace, která zobrazuje teoreticky popsané skutečnosti. Jedná se o diář s databází v xml. Při spuštění vypíše aplikace aktuální poznámky ze souboru na daný a následující den a nabídku akcí.

3 Literární rešerše

3.1 Základní programovací paradigma

V této kapitole jsou popsána jednotlivá využívaná programovací paradigmata. Ta se vyvíjejí a různé programovací jazyky poskytují podporu pro jedno, či několik paradigmat.

Při programování závisí vnitřní struktura programu na použitém programovacím jazyce, který programátora nutí vytvářet programy se strukturou, jaká odpovídá danému paradigmatu. Paradigmata lze podle Skoupila rozdělit do několika skupin. Každá skupina je reprezentována svým typickým programovacím jazykem [8].

3.1.1 Skupiny paradigmat

- Naivní paradigma
- Procedurální paradigma
- Objektově orientované paradigma
- Funkcionální paradigma
- Logické paradigma
- Speciální paradigma
    - Paralelní a distribuované programování
    - Datové programování
    - Webové programování
    - Textové programování

Naivní paradigma

Toto programovací paradigma se vyznačuje, podle Skoupila [8], především velkou chaotičností a nekoncepčností. Programovací jazyk, který podporuje toto paradigma, je charakterizován tím, že neumožňuje modularitu, poskytuje minimální podporu pro datovou abstrakci a je minimálně strukturovaný. Typickým příkladem takovéhoto programovacího jazyka je jazyk BASIC, který často využívá konstrukci skoku (pomocí příkazu GOTO) pro přechod na jiný programovací řádek.

Procedurální paradigma

Procedurální paradigma je, podle Skoupila [8], někdy nazýváno *klasické* nebo *imperativní*. Toto paradigma má velmi univerzální použití a je velmi rozšířené zejména pro malé projekty v komerční sféře. Jedná se o jedno z nejstarších paradigmat (jazyk FORTRAN byl vytvořen v roce 1954) jehož základní úlohu hrají příkazy. Průběh výpočtů je dán sekvencí

po sobě jdoucích příkazů. Významné využití zde mají také cykly. Zástupci programovacích jazyků, kteří využívají procedurální paradigma, jsou například zmíněný Fortran. Dále se jedná o Pascal nebo jazyk C [8].

Objektově orientované paradigma

Použití tohoto paradigmatu bylo původně v oblasti simulace a umělé inteligence, ale v současnosti má již univerzální použití a to například pro rozsáhlé projekty.

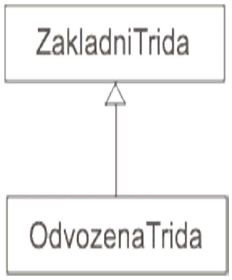
Za základní programovací prvky jsou v tomto paradigmatu považovány objekty. Tyto objekty reprezentují objekty reálného světa (předměty, živočichy, události). Každý objekt je nositelem informací o svém stavu a má možnost tento stav změnit. Objekt je nositelem dat, která v sobě zapouzdřuje, ale také procedur, které s těmito daty pracují a jejichž průběh výpočtu je určen zasíláním zpráv mezi jednotlivými objekty. Díky těmto zprávám je objektům sděleno, zda a jak mají změnit svůj stav. Typickými zástupci jsou Smalltalk, C++, C #, Java.

Objektově orientované programování

Pecinovský ve své knize [7] píše, že objektový přístup k programování je v dnešní době již zaběhlý způsob návrhu aplikací. Základním paradigmatem objektově orientovaného programování (zkrácené OOP z anglického Object-Oriented Programming) je používání *objektů*. Tyto objekty nejsou určeny jen množinou dat, ale také množinou operací, které jsou přesně stanovené a definované a které smí objekt provádět. Objekty mezi sebou mohou komunikovat prostřednictvím zpráv [7].

Základní vlastnosti Objektově orientovaného programování

- Zapouzdření (Encapsulation)
  - Základním pojmem OOP je *class*, neboli třída. Tato třída v sobě zahrnuje data a operace. Konkrétní data ve třídě se nazývají *instance třídy*. Například třída nese pojmenování Osoba a instancí je Novák.
- Dědičnost (Inheritance)
  - V OOP se tak nazývá možnost odvozovat nové třídy, které dědí ze své nadtřídy data a metody. V těchto odvozených třídách je možné přidávat nové metody a data.
  - V Javě není umožněna vícenásobná dědičnost, lze tedy dědit pouze z jedné základní třídy do jedné odvozené třídy.



Obrázek 1 Dědičnost,
zdroj:[http://projektysipvz.gytool.cz]

- Polymorfismus
  - Umožňuje pomocí jednoho příkazu, neboli zprávy, zpracovávat více podobných objektů. Podstatou je metoda nebo metody, které mají všichni potomci definované stejnou hlavičkou, ale jiným tělem. Pro vysvětlení se využívá obrázek 2, na kterém mají všechna zvířata v rozhraní metodu *Speak(),* ale každé ji vykoná po svém [10, 11].

Obrázek 2 Polymorfismus [Zdroj: http://www.devbook.cz]

Funkcionální paradigma

Typickým zástupcem programovacích jazyků využívajících funkcionální paradigma je Lisp nebo Scheme. Toto paradigma nachází své využití především ve výuce a výzkumu. Průběh výpočtu je ve funkcionálním paradigmatu založen na postupném aplikování funkcí. Velmi využívaným nástrojem je zde rekurze a funkce vysoké úrovně [8].

Logické paradigma

V tomto paradigmatu je program tvořen množinou faktů a pravidel a takto je předkládán počítači. Průběh výpočtu je založen na metodě unifikace a zpětného řetězení. Jedná se o princip, při kterém je předloženo tvrzení a systém se na základě faktů snaží prokázat, že se jedná o tvrzení pravdivé. Využití toto paradigma nachází zejména v oblasti znalostních systémů. Zástupcem z oblasti programovacích jazyků je jazyk Prolog.

Speciální paradigma

Je možné se setkat i s jazyky, které vycházejí z určitého paradigmatu, ale přidávají k němu navíc podporu pro další činnost. Dále jsou uvedeny příklady skupin programovacích jazyků, podle Skoupila [8], které tvoří vlastní paradigma.

- Paralelní a distribuované programování
    - S vývojem a využíváním víceprocesorových operačních systémů bylo nutné vyvinout jazyky, které umožňují efektivní paralelizaci a distribuci. Paralelizací se rozumí dekompozice programu na jednotlivé úlohy, které mohou být paralelně zpracovávány různými procesory sdílejícími stejnou paměť v rámci operačního systému. Distribuce je také dekompozice programu na úkoly, ale tyto úlohy mohou být současně zpracovávány různými počítači.
- Datové programování
    - Jedná se o jazyky vhodné pro manipulaci s velkými objemy dat, jejich snadnou manipulací a dále zajišťují přímé připojení k databázím. Příkladem programovacího jazyka v datovém programování je jazyk COBOL.
- Webové programování
    - Toto paradigma zahrnuje programovací jazyky pro tvorbu webových aplikací. Jako zástupce lze uvést například jazyk PHP [8].

3.2 Java

V této kapitole jsou uvedeny základní informace o programovacím jazyku Java, který je použit v praktické části práce pro tvorbu aplikací.

3.2.1 Historie

V roce 1991 byla vytvořena skupina vývojářů společnosti Sun, pod vedením Jamese Goslinga a Patrika Naughtona. Dala si za cíl vytvořit programovací jazyk pro spotřebitelské zařízení. Cílem bylo vytvořit jazyk, který by nebyl závislý na platformě. Projekt byl pojmenován kódovým jménem Green.

Nejprve byl vzkříšen pro tento účel programovací jazyk USCD Pascal, za jehož zrodem stojí Niklaus Wirth. Tento jazyk ovšem nebyl shledán vhodným. Nakonec navrhl nový jazyk James Gosling a pojmenoval ho Oak. Za přímého předchůdce lze považovat jazyk C++. Ovšem po zjištění, že již jeden programovací jazyk se stejným názvem Oak existuje, musel být Goslingům nový programovací jazyk přejmenován na Java [12].

V roce 1993 si firma Sun, podle Herouta [1], uvědomila vzrůstající roli WWW a naskýtající se možnost, využít Javu pro programování aplikací určených právě pro WWW. Poprvé došlo k představení Javy na konferenci SunWorld v roce 1995. Již v té chvíli bylo zřejmé, že tento programovací jazyk bude v budoucnu hrát významnou úlohu při programování webových aplikací. Zájem o toto využití Javy byl zapříčiněn zejména díky začínající vlně zájmu zejména o obchodní využití WWW. Technologie byla licencována společnostmi IBM, Symantec, Inprise a Microsoft. V současnosti je nejnovější verzí Java 7. Stále však dochází k mnoha updatům [1].

3.2.2 Java timeline

V následující části jsou popsány jednotlivé verze Javy a jejich nový přínos.

1995

Přejmenování programovacího jazyka Oak na programovací jazyk Java z důvodu existence jiného programovacího jazyka s názvem Oak. Toto přejmenování nastalo poté, co bylo rozhodnuto vývojáři z Oak o registraci ochranné známky. V tomto prvním vydání se celá Java vešla na jednu disketu [2].

1997

V tomto roce vznikla verze 1.1, která zavedla vnitřní a vnořené třídy. Dále byly některé třídy výrazně upraveny a počet veřejných tříd se oproti první verzi více než zdvojnásobil, a to na 477.

1998

Zlomová v historii vývoje jazyka Java se, podle Pecinovského [2], považuje verze 1.2. Tato verze byla vydána na přelomu let 1998 a 1999. V této verzi došlo k zásadnímu přepracování celé knihovny a zavedení mnoha dalších. Díky tomu vzrostl počet veřejných tříd na 1524. Tyto změny koncepce práce s jazykem byly považovány za natolik zásadní, že nová verze platformy a jazyka dostala označení Java 2 [2].

2000

Verze 1.3, nebo přesněji Java 2 verze 1.3, která se objevila v tomto roce, pokračovala v evoluci a výsledkem bylo rozšíření knihovny na 1840 veřejných tříd.

2002

Verze 1.4 po Javě 1.1 přinesla rozšíření syntaxe. V této verzi zavedla klíčové slovo *assert*, které velmi usnadnilo ověřování bezchybnosti programu a vyhledávání chyb v instalovaných programech. Dále také nastalo zásadní rozšíření knihovny. Ta nyní obsahovala 2723 veřejných tříd.

2004

Na podzim roku 2004 vychází verze Javy z počátku označovaná jako 1.5, která ale byla později firmou Sun přejmenována na Java 2 verze 5.0. Z tohoto důvodu se o ní hovoří jako o verzi 5.0. V této verzi došlo k rozšíření knihovny na 3270 veřejných tříd. Počet všech tříd, včetně interních a pomocných, přesahuje 15 000. Podstatné bylo zásadní rozšíření syntaxe jazyka, díky čemuž došlo ke změně číslování verze [2].

2006

Java 6 přináší nový koncept profilů. Profil je podmnožina Java EE, která může být doplněna o další technologie pro plnění určité úlohy. Konkrétním příkladem je *Webprofile*, sloužící pro provozování webových aplikací, který nezahlcuje server věcmi, které pro provoz webové aplikace nepotřebuje.

2011

V novinkách, které jsou v JDK 7, byla zahrnuta například část projektu *Coin* (změna syntaxe a sémantiky jazyka). Jedná se o projekt, který vnáší změny syntaxe a sémantiky. V rámci tohoto projektu nedošlo k přidání nových instrukcí, ani na změnu bajt kódu o další instrukce. Novými prvky, které *Coin* přinesl, jsou například deklarace celočíselných binárních konstant, příkazy *swich* a *řetězce* a dále využití podtržítka pro zlepšení čitelnosti numerických konstant [13].

2014

Na tento rok je po posunu termínu naplánováno vydání verze Java JDK 8. Důvodem opoždění je vytížení vývojářů opravou bezpečnostních chyb. Dále nová verze bude zahrnovat lambda funkce. Plánovaný datum je počátek roku 2014 [14].

3.2.3 Aplikace vs Applet

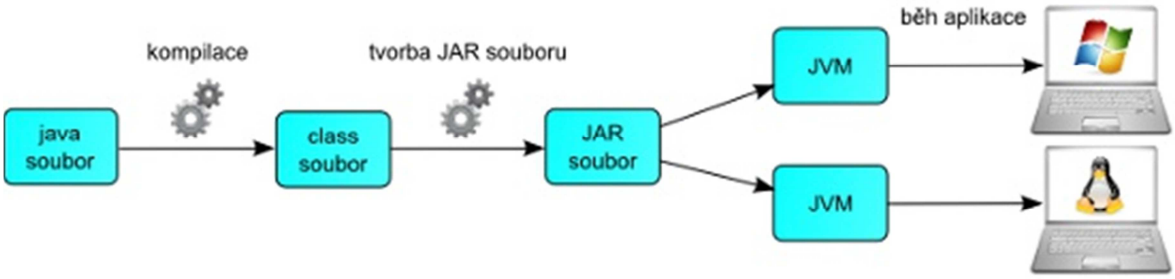
- Applet
    - Program určený pro umístění na WWW server, kde je včleněn do HTML dokumentu tvořícího WWW stránku. Při zobrazení této stránky Java-kompatibilním prohlížečem se Applet automaticky nahraje do klientského počítače, kde je následně spuštěn.
- Aplikace
    - Samostatný program, který vyžaduje pro svůj běh Java Platformu, nikoliv prohlížeč jako applet. Na aplikaci obecně nejsou kladena bezpečnostní omezení [15].

3.2.4 Zpracování programu v Javě

V Javě prochází program pěti fázemi. Tyto fáze jsou v knize [1] pojmenovány jako editování, překlad (kompilace), zavedení, ověřování (verifikace) a prováděním. Ve většině programovacích jazyků jsou čtyři z těchto pěti fází zcela běžné. V Javě je fáze ověřování něco nového, ale zejména pro programování WWW, velmi důležitého. Tato fáze umožní dosáhnout velmi vysoké bezpečnosti spuštěného programu a to díky ochraně toho, kdo program spustí [1].

Překlad v jazyce Java neprobíhá přímo do spustitelného souboru pro jednotlivé operační systémy, ale do pseudojazyka nazývaného byte-code. Díky tomu, že tento jazyk

není závislý na cílovém počítači, je docíleno toho, že programátora nemusí zajímat, na jakém cílovém počítači jeho program poběží. Bytecode je uložen v souboru s příponou .class. Tento soubor je pak dále zaváděn z disku počítače do paměti a přitom je provedeno ověření byte-codu, které je díky jeho nezávislosti na platformě prováděno jednotně. Po ověření je program spouštěn pomocí interpretu [1].



Obrázek 3 Zpracování programu v Javě [Zdroj: http://www.java-skoleni.cz/info/java.php]

3.2.5 Java Runtime Environment (JRE)

Rozhraní nutné pro běh programů, které potřebují pro svoji funkci Java rozhraní. Tyto applety se velmi často vyskytují na webových stránkách. JRE zahrnuje interpret Javy *java* a standardní knihovny.

Nejdůležitější částí Java technologie je vlastní prostředí, zastoupené na dané platformě virtuálním strojem. Tento virtuální stroj se nazývá Java Virtual Machine, zkráceně JVM. Skládá se z runtime systému a interpretu. Runtime systém realizuje vazbu na hardware a interpret vykonává bytecode.

Pro urychlení je možnost volitelně nahradit interpret JIT (Just-In-Time) kompilátorem. Ten při běhu programu nejprve provádí překlad do strojového kódu daného procesoru.

JRE je samostatně distribuovaná součást JDK. Hlavní výhodou je snadná instalace, která je prováděna pomocí jednoho instalačního souboru, bez nutnosti nestandardních úkonů během instalace. Stažení JRE je možné přímo ze stránek oracle.com [5, 15, 16].

3.2.6 Java Development Kit (JDK)

Volně dostupný balík vývojových programů, který obsahuje JRE, překladač zdrojového kódu do bytecodu *javac*, debugger a další vývojové nástroje.

Číslování verzi JRE a JDK je shodné a má tento význam:

```
JDK 1.4.2
    | | |
    | | +--- bug fix number (větši číslo -> méně chyb)
    | +----- minor verze (mění se, dojde-li ke změnám
    |        či rozšířením v jazyce nebo knihovnách)
    +------- major verze (interně zatím vždy 1)
```

Obrázek 4 Verze JRE a JDK [Zdroj: http://v1.dione.zcu.cz/java/uvod.html]

JDK se dále dělí na dílčí platformy, které obsahují specifičtější knihovny a komponenty než základní verze. Tyto platformy jsou Java Card, Java Micro edition, Java Standard edition a Java Enterprise Edition.

Technologie Java Card umožňuje vývojářům vytvářet, testovat a nasazovat aplikace a služby, rychle a bezpečně. Tento zrychlený proces snižuje náklady na vývoj, zvyšuje diferenciaci výrobků a zvyšuje hodnotu pro zákazníky. Technologie Java Card umožňuje snadno integrovat prvky zabezpečení na kompletní řešení softwaru Java [17].

Jádrem platformy Java Mobile je Java Platform, Micro Edition (Java ME). Java ME nabízí robustní a flexibilní prostředí pro aplikace běžící na mobilních a dalších vestavěných zařízení, jako jsou například mobilní telefony, televizní set-top boxy, e-čtečky, Blu-Ray čtečky, tiskárny a další [18].

Java Platform, Standard Edition (Java SE) umožňuje vyvíjet a implementovat aplikace Java do počítačů a serverů. Java nabízí bohaté uživatelské rozhraní, výkon, všestrannost, mobilitu a bezpečnost, což dnešní aplikace vyžadují [19].

Java EE byl zpočátku vyvinut pro implementace podnikových aplikací. Je to platforma, která se zaměřila na robustnost, Web Services a snadné nasazení. Zajišťuje zpětnou vazbou prostřednictvím Java Community Process (JCP). Java EE představuje univerzální standard v podniku IT, usnadňující vývoj, implementaci a správu vícevrstvých, server-centric aplikací [3].

Pro vývoj aplikací v Javě stačí programátorovi jakýkoliv běžný textový editor, tak ale v dnešní době nebude již nikdo pracovat. V praxi se využívají integrovaná vývojová prostředí, zkráceně IDE, ze kterých je nejpoužívanější vývojové prostředí Eclipse. Z ostatních prostředí je nutné zmínit NetBeans, jenž má kořeny v České republice a IntelliJ Idea. Idea je příklad jednoho z mála placených vývojových prostředí [20].

Obrázek 5 Job Trends [Zdroj: http://www.java-skoleni.cz/info/java.php]

3.2.7 Bezpečnost v Javě

Jednou z nejvíce zdůrazňovaných předností programovacího jazyka Java je, podle Pavlíčkové a Pavlíčka [4], bezesporu jeho bezpečnost. Základní bezpečnostní prvky jsou již implementovány přímo v jazyce, avšak je velmi důležité tyto prvky inicializovat a především správně nastavit tak, aby mohli být účinně využity.

Kromě těchto prvků jsou v Javě k dispozici další knihovny, které zaručují zvýšení bezpečnosti aplikací. Především se jedná o nárůst bezpečnosti při síťové komunikaci a víceuživatelském přístupu. Označováno je toto opatření jako Java security a je tvořené třemi balíčky pojmenovanými JAAS, JCE a JSSE [4].

- JAAS (Java Authentication and Autorization Service)
    - Tento balíček poskytuje rozhraní, které zajišťuje ověřování a autorizaci uživatelů, kteří využívají jednotlivé aplikace.

- JCE (Java Cryptography Extension)
    - Balíček pro zajištění šifrované komunikace, digitální podpisy atd.

- JSSE (Java Secure Sockets Extension)
    - Rozhraní, které umožňuje programování aplikací s přenosem po zabezpečené vrstvě SSL.

Vlastnosti jazyka Java, jako je například typová kontrola, povinnost ošetřovat synchronní výjimky, nebo správa paměti, zajišťují bezpečnost jazyka. Další důležitou složkou zabezpečení je možnost kontrolovat, odkud může aplikace prostřednictvím classloaderu nahrávat příslušné soubory typu class a co může kód z rozlišných zdrojů dělat.

Na počátku vývoje byla Java navržena tak, aby rozeznávala dva druhy kódu. Konkrétně se jednalo o kód lokální, neboli kód uložený a spouštěný lokálně a kód vzdálený, který byl uložen na síti a poté stažen a spouštěn lokálně. Rozdíl byl v tom, že lokálnímu kódu, jakožto kódu důvěryhodnému bylo umožněno vše. Naopak vzdálený kód měl velmi omezený přístup k prostředkům a to pouze v rámci sandboxu [21].

Sandbox je bezpečnostní mechanismus, který slouží pro spouštění programů z nedůvěryhodných zdrojů. Kontrolu důvěryhodnosti provádí třída SecurityManager. Od verze 1.2 je možné pro kód z různých zdrojů možné nastavit specifické možnosti přístupu ke zdrojům systému. Pracuje se nad tzv. ochrannými doménami. Tato doména je definovaná pomocí security policy. Sandbox je typem domény s pevně stanoveným ohraničením.

Pokud je spuštěn kód, který je součástí SDK (Software Development Kit) je automaticky považován za důvěryhodný a s tím je spojeno, že má nastavena veškerá oprávnění. Aplikační kód je standardně spouštěn bez pomoci správce zabezpečení. Pokud je nutné aplikaci nastavit práva, musí být k aplikaci přiřazen soubor zásad zabezpečení, ve kterém je nadefinováno, co je aplikaci povoleno. Pokud je v prohlížeči spuštěna stránka obsahující applet, je správce zabezpečení spuštěn automaticky a appletu je přiřazen příslušný sandbox [4, 21].

3.3 Úvod do teorie Javy

Programovací jazyk Java využívá kódování Unicode, které umožňuje využívat znaky většiny národních abeced. V této kapitole následuje úvod do základních syntaxí, které jsou využité v programech, které slouží pro popis prostředí Eclipse.

Unicode je průmyslový výpočetní standard navržený tak, aby umožnil důsledně a jednoznačně kódovat znaky využívané v jazycích po celém světě. Standard Unicode používá hexadecimální vyjádření znaků. Například, hodnota 0x0041 představuje latinské písmeno A. Standard Unicode byl původně navržen pomocí 16 bitů pro zakódování znaků, protože primární stroje byli 16-bitové počítače [22].

3.3.1 Komentáře

Jako většina programovacích jazyků má i Java možnost vkládat do zdrojového kódu programu komentáře. Tyto komentáře slouží, podle Herouta [1], k popsání kódu a jeho snadnému pochopení ostatními lidmi, kteří se s kódem dostanou do styku. Komentář by měl být vždy na místě, kde se nachází neobvyklý programátorský obrat. Dále je vhodné umístit při tvorbě programu komentáře na místa, kde se vyskytl nějaký problém během programování, aby se programátorovi nestalo, že po návratu k danému programu bude dlouze vzpomínat, kde se chyba stala [1].

Java nabízí programátorům tři typy komentářů. Jedná se o jednořádkový komentář, komentářový blok a dokumentační komentář.

Jednořádkový komentář začíná znakem // a vše za tímto znakem až do konce dané řádky je automaticky bráno jako komentář. Například lze uvést deklaraci proměnné s komentářem: private String text; // definování proměnné text jako String.

Komentářový blok začíná znaky /* a je brán za komentář až do ukončovacího znaku */. Pomocí takovéhoto bloku lze použít rozsáhlejší komentář, který zabírá více než jednu řádku. Komentovat takto jednořádkově je samozřejmě taky možné. Příkladem komentářového bloku může být:

```
/* V následující třídě jsou uvedeny deklarace proměnných, se
kterými se bude v tomto programu pracovat*/ public class Notes {… [Zdroj:
Autor]
```

Jako poslední možnost nabízí Java dokumentační komentář. Jeho využití je pro automatické generování dokumentace pomocí programu javadoc.exe. Komentáře jsou definovány pomocí počátečních znaků /** a jsou zakončeny znaky */. Dále se v těchto komentářích využívají upřesňující znaky, které jsou reprezentovány symbolem @. Je zde možné využít i veškeré znaky z formátu HTML, jelikož výstupem programu javadoc.exe je HTML formát. Příkladem dokumentačního komentáře může být například zapsání autora programu [1].

```
/**
 * @author Jan Kuník
 *
 */.[Zdroj: Autor]
```

3.3.2 Způsoby zápisu identifikátorů

Programovací jazyk Java důsledně rozlišuje malá a velká písmena a často se programátor setká s tím, že dva identifikátory jsou rozlišeny pouze velikostí písmen. Typickým příkladem, který bude dále rozebrán, je zápis proměnné a třídy. Proměnné se v Javě vždy zapisují pomocí malých písmen, například *datum*. Třídy jsou vždy charakteristicky velkým počátečním písmenem, například *Osoba* [1].

Délka identifikátorů není omezena, musí ovšem každý začínat písmenem, nebo podtržítkem. Dále mohou již následovat i číslice a znak "$". Pro zapsání více slovných identifikátorů je vhodné využít například podtržítko (*datum_narozeni*), nebo zapsat bez mezer s velkým písmenem (*datumNarozeni*). Pro zápis nelze použít mezeru, proto proměnnou *datum narození* nelze využít.

Striktně dodržované konvence pro zápis identifikátorů představují dle Herouta [1]:

- Třídy a rozhraní
    - Identifikátor musí, jak již bylo na příkladu ukázáno, začínat velkým písmenem a zbylá písmena jsou malá. Jediná výjimka nastává v případě, že třída je víceslovně pojmenována. Například třída StringBuffer, která se využívá ke změnám řetězců.

- Metody a proměnné
    - Zde platí, že identifikátor začíná malým písmenem a stejně jako u tříd při víceslovném pojmenování, je za začátek dalšího slova považováno velké písmeno. Metody jsou od proměnných rozpoznatelné pomocí závorek na konci názvu. Příkladem může být proměnná *dateTime a metoda getDateTime().*
- Balíky (package)
    - Identifikátor je složen pouze z malých písmen. Z tohoto důvodu je ve víceslovných názvech využito oddělování jednotlivých slov tečkou. Na následujícím příkladu je dobře vidět oddělení pomocí tečky: package cz.culs.pef.kii.java.priklad.
- Konstanty
    - V konstantách jsou využity pouze velká písmena a ve víceslovných názvech konstant je z tohoto důvodu využito podtržítko. Například konstanta MAX_VALUE.

Jako identifikátor nesmí být v programovacím jazyce Java užito čtyřicet osm rezervovaných slov. Dále je uveden seznam těchto rezervovaných slov [1, 23].

abstrakt,	continue,	import,	public, return, short,
boolean,	default, do,	instanceof, int,	static, strictfp,
break,	double, else,	interface, long,	super, switch,
byte, case,	extends, final,	native, new,	synchronized, this,
catch,	finally, float,	package,	throw, throws,
char,	for, if,	private,	transient, try, void,
class,	implements,	protected,	volatile, while.

Dále slova *const* a *goto* nejsou používány jako klíčová slova, ale nesmí se také využívat jako identifikátory.

3.3.3 Metoda main

Hlavní metoda *main* je metoda, která je vyvolána při spuštění programu jako první. Každý program musí nutně tuto metodu obsahovat a zároveň metoda musí být zapouzdřena ve třídě. Pro tuto metodu platí ještě několik pravidel [1].

Třída *main,* by se měla nacházet ve třídě, která je typu *public,* neboli veřejná. Jako další podmínku je nutné dodržet striktní zápis. Tento zápis má následující podobu:

```
Public static void main (String[] args){
//Tělo metody main
}[Zdroj: Autor]
```

Jako důsledek nedodržení některého ze zmíněných pravidel je to, že program nebude možné přeložit [5].

3.3.4 Základní datové typy

Základní datové typy jsou někdy též nazývány primitivní datové typy (*primitive data types)* a jsou v Javě zastoupeny několika druhy. Jedná se o znakové, logické, celočíselné, reálné a prázdný datový typ void. Datový typ *void* se využívá pouze u funkcí k určení toho, že funkce nemají žádnou návratovou hodnotu. Java definuje u každého datového typu počet bajtů, které datový typ zabírá v paměti [1].

Celočíselné datové typy

V Javě existují čtyři druhy celočíselných datových typů. Liší se od sebe pouze svojí velikostí a s ní spojeným rozsahem zobrazovaných čísel. V následující tabulce je zobrazen přehled datových typů.

Název	Bitů	Rozsah	Řád
byte	8	-128 až +127	10^2
short	16	-32768 až 32767	10^4
int	32	-2 147 483 648 až 2 147 483 647	10^9
long	64	-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807	10^{18}

Tabulka 1 Celočíselné datové typy [Zdroj: 1, s. 36]

Znakový typ a jeho konstanty

Znakový typ je v programovacím jazyce Java pouze jeden. Jedná se o znakový typ *char* a má velikost 16 bitů (2 bajty). Je to z důvodu toho, že Java pracuje se znaky Unicode.

Znakové konstanty jsou vždy uzavírány do apostrofů a mohou být reprezentovány několika způsoby.

- Posloupností "\uXXXX "
    - V tomto případě písmena X představují šestnáctkové číslice. Tento zápis se využívá v případě akcentovaných znaků, ale lze tak zapsat jakýkoliv znak. Jako příklad lze uvést" \u00F3", což je posloupnost pro písmeno ó.
- „*Escape“* sekvencí
    - Tento typ zápisů se využívá například pro odřádkování textového výpisu, kdy se do výstupu začlení výraz " \n". Konkrétním příkladem může být System.out.print("Odřádkování \n"). Další sekvence jsou například "\t" pro tabulátor, "\r" pro návrat na začátek řádky a "\b" pro posun doleva.
- Osmičkovým zápisem
    - Jedná se o zápis ve tvaru "\ooo", kde „o“ značí osmičkovou číslici. Vždy je nutné zapsat všechny tři a to i v případě, že se jedná o nevýznamovou nulu například "\007" [1].

Řetězcové konstanty

Způsob tvoření řetězcových konstant je stejný jako v případě konstant znakových, jediná změna je, že řetězcové konstanty jsou uzavřeny do závorek. V jednom řetězci lze libovolně kombinovat výše zmíněné způsoby zápisu znaků. Například slovo „končí“ lze zapsat jako "kon\u010D\u00ED".

Logický typ a jeho konstanty

Jako logický typ se v Javě využívá typ *boolean*. Tento typ může nabývat pouze dvou hodnot a tyto hodnoty jsou představovány logickými konstantami *true* (logická 1) a *false* (logická 0). Lze pracovat i s negací vyjádřenou pomocí "!", logickým součinem "&&" a logickým součtem, neboli disjunkcí vyjádřenou pomocí "||".

Reálné datové typy a jejich konstanty

Programovací jazyk Java rozeznává dva reálné datové typy a to *float* a *double.* Oba vyhovují mezinárodnímu standardu, což znamená, že v Javě mají stejné zobrazení jako

v jiných programovacích jazycích. Příkladem reálné konstanty jsou čísla s desetinou čárkou.

Při operacích s reálnými čísly, může nastat situace, že výsledek bude nabývat hodnot nekonečna (kladného nebo záporného). Tyto konstanty jsou POSITIVE_INFINITY a NEGATIVE_INFINITY. Tyto hodnoty se mohou objevit, pokud se dělí nulou. Další speciální hodnota je NaN (Not a Number). Tato hodnota vzniká při dělení nuly nulou. Tyto hodnoty lze otestovat pomocí metod isInfinite() a isNan() [1].

3.3.5 Pole

Pole se využívají pro ukládání více proměnných stejného typu. Pole si lze, podle Čápka [24], představit jako množství „zásuvek“ kde v každé je uložený jeden prvek pole a tyto „zásuvky“ jsou očíslovány pomocí indexu. Zejména ve starších programovacích jazycích nebylo možné vytvářet dynamická pole za běhu programu, protože pole se muselo deklarovat přímo ve zdrojovém kódu s danou velikostí. Pole je jednoduchá a klíčová struktura v jazyce Java. Rychle se s ním pracuje, především z toho důvodu, že prvky jsou v paměti uloženy za sebou a zabírají všechny stejně místa. Díky tomuto uspořádání se k nim rychle přistupuje. Mnoho vnitřních funkčností v Javě pracuje s polem nebo pole vrací [24].

Pro deklaraci proměnné, která je typu pole a uchovává v sobě položky typu *integer,* se využívá následující zápis: *int [] x;* Z tohoto zápisu je patrné, že pro deklaraci pole jsou v Javě využity hranaté závorky. X je pouze proměnná, kterou je deklarováno, že v ní bude uloženo pole *integerů*. Pro založení pole je nutné využít klíčové slovo *new*. Při zapsání kódu: *int[] x = new int[10];* dosadíme do proměnné *x* pole o velikosti deseti *integerů* [1].

Pro naplnění pole se využívá toho, že na každý prvek pole je odkázáno pomocí indexu. Důležité je to, že první prvek pole má index o hodnotě 0. To znamená, že pokud je pole o velikosti deseti prvků, tak desátý prvek má hodnotu indexu rovnu číslu devět. Naplnění je vidět na následujícím příkladu.

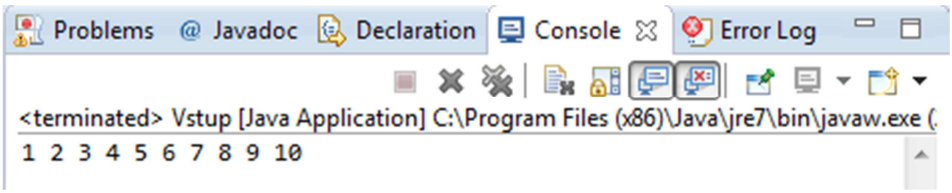
```
public static void main(String[] args) {
            int[]  x = new int[10];
            x[0] = 1;
[Zdroj: Autor]
```

Z kódu je patrné, že do prvního prvku pole byla přiřazena hodnota jedna. Tuto hodnotu je možné vypsat pomocí příkazu *System.out.print(x[0]);*. Naplňovat pole takto je

velmi zdlouhavé a neefektivní, proto se k plnění polí využívají cykly *for*. Toto využití cyklu je vidět na následujícím příkladu.

```
public static void main(String[] args) {
            int[] x = new int[10];        \\ vytvoření pole
            for (int i = 0; i < 10; i++)
                    x[i] = i + 1;         \\ naplnění prvků čísli
            for (int i = 0; i < x.length; i++)
              System.out.printf("%d ", x[i]); \\ výpis hodnot
            }
            [Zdroj: Autor]
```

V tomto cyklu se nejprve vytvoří pole typu *int* o velikosti deseti prvků, poté následuje cyklus *for,* který postupně naplňuje každý prvek hodnotou zvýšenou o jedna oproti předchozímu. Druhý cyklus *for* slouží k výpisu celého pole. Na obrázku 6 je vidět výpis všech prvků pole.



Obrázek 6 Výpis pole [Zdroj: Autor]

Třída Arrays a její metody

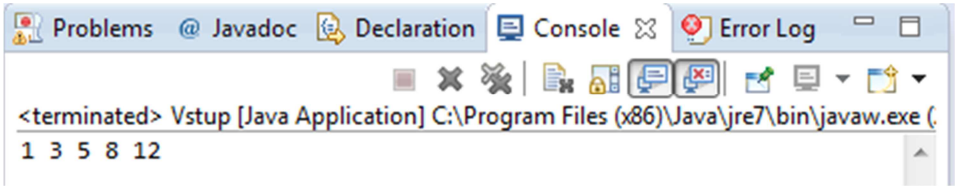
V programovacím jazyce Java je možné využít metod ze třídy *Arrays*. Tyto metody popisuje Lowe v [5] jako metody sloužící programátorovi pro práci s polem a je nutné je nejdříve naimportovat. Import se provede zapsáním *import java.util.Arrays;*[5].

Jako první metoda je uvedena metoda *sort ().* Tato metoda umožňuje setřídit dané pole. Jako parametr je nutné uvést jen pole, které má být setříděno. Automaticky třídí pole podle toho, o jaké pole se jedná. Pole typu *string* setřídí podle abecedy a pole *int* podle velikosti. Názorně je to patrné na následujícím příkladu.

```
public static void main(String[] args) {
            int[] x = new int[5]; \\ deklarace pole o velikosti
pěti prvků

            x[0] = 1; \\ naplnění pole
            x[1] = 3;
            x[2] = 12;
            x[3] = 8;
            x[4] = 5;
        Arrays.sort(x); \\ volání funkce, která pole setřídí
        for (int i = 0; i < x.length; i++)
```

```
            System.out.printf("%d ", x[i]); \\ výpis pole
        }[Zdroj: Autor]
```



Obrázek 7 Setřídění pole dle velikosti

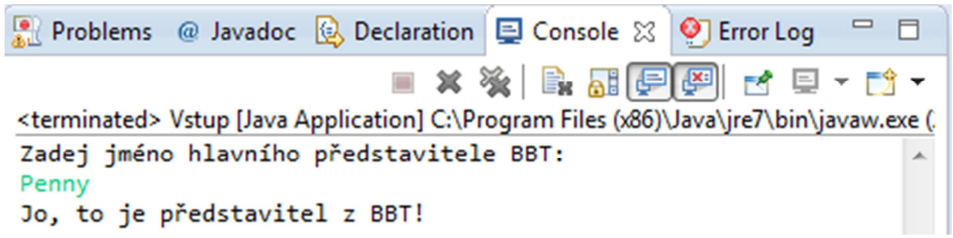
Z výpisu je patrné, že i když bylo pole nadefinováno jako [1, 3, 12, 8, 5], tak po zavolání funkce *sort(),* se na výstupu zobrazí seřazené dle velikosti.

Další důležitou metodou v třídě *Arrays*, kterou Lowe [5] popisuje, je metodu *binarySearch().* Tato metoda nám umožní na setříděném poli vyhledávat prvky. Metoda vrací index prvního nebo posledního nalezeného prvku. V případě nenalezení prvku vrátí hodnotu -1. Metoda využívá dva parametry. Prvním je pole, ve kterém má hledat a druhým hledaný prvek [5].

```
      import java.util.Arrays; \\ import potřebných nástrojů a tříd
      import java.util.Scanner;
      public class Pole{
      public static void main(String[] args) {
            Scanner sc = new Scanner(System.in, "Windows-1250");
            String[] bbt = {"Leonard", "Sheldon", "Penny", "Howard",
"Rajesh"};
      \\ definování pole
      System.out.println("Zadej jméno hlavního představitele BBT: ");
      String postava = sc.nextLine(); \\ načtení zadání uživatele

      Arrays.sort(bbt); \\ setřídění pole
      int pozice = Arrays.binarySearch(bbt, postava); \\ vyhledání
            if (pozice >= 0)
                  System.out.println("Jo, to je postava z BBT!");
            else
                  System.out.println("Tato postava v BBT není");

            }}
            [Zdroj: Autor]
```



Obrázek 8 Vyhledání v poli [Zdroj: Autor]

Na příkladu je zřejmé, že po naplnění pole členy seriálu The big bang theory a jeho setřídění, je uživateli vypsáno, aby zadal jméno seriálového představitele. Pokud zadá jméno, které je v seznamu uvedeno, vypíše se do konzole zpráva: Jo, to je postava z BBT!. Pokud, by uživatel zadal cokoliv jiného, objeví se zpráva: Tato postava v BBT není. Důležité je to, že uživatel musí zadat jméno přesně tak, jak je v poli nadefinováno. Pokud by zadal například jméno s malým počátečním písmenem, objeví se již druhý výpis.

ArrayList

Jedná se o kolekci (struktura, do které lze ukládat více objektů, například pole), kterou lze chápat jako nadstavbu pole. Při využívání *arrayListu* není potřeba předem definovat, jak velký bude. Sám se přizpůsobí na počet prvků, které v něm budou uloženy. V tomto *arrayListu* lze za běhu programu přidávat a mazat jednotlivé záznamy. Lze si ho představit jako databázi. Při deklaraci *arrayListu* je nutné uvést, s jakým typem proměnných se bude pracovat.

ArrayList se deklaruje následovně: *ArrayList<Integer> cisla = new ArrayList<Integer>();*. Takto deklarovaný *ArrayList* by pracoval s proměnnými typu *integer*. Pro práci je nutné ještě provést import *import java.util.ArrayList*. Konkrétní příklad bude uveden později v praktické části práce při popisu prostředí.

Metod v kolekci *ArrayList* je mnoho, především oproti poli. Jednou z velkých výhod je přidávání a mazání prvků. Dále jsou uvedeny některé metody s popisem jejich využití [5, 24].

- size() – Má stejné použití jako *length* na poli. Po zavolání vrací počet prvků v kolekci.
- add(položka) – Metoda sloužící pro přidání prvku do listu.
- addAll(kolekce) - Přidá do listu více položek, např. z pole.
- clear() - Vymaže všechny položky v listu.
- contains(položka) – Vrací logickou hodnotu true/false v závislosti na tom, zda *ArrayList* obsahuje zadanou položku.
- remove(položka) - Vymaže první nalezenou položku.
- removeAll(index, počet) - Vymaže daný počet prvků od daného indexu [25].

Deklarace proměnných

Pojmem deklarace je myšlen příkaz, který určité proměnné určitého typu přidělí jméno, paměťový prostor a počáteční hodnotu. Pokud není inicializační hodnota uvedena, proběhne implicitní inicializace, při které je proměnné přidělena počáteční hodnota vzhledem k tomu, o jaký datový typ proměnné se jedná. Konkrétně celočíselné typy jsou nastaveny na hodnotu 0, reálné na hodnotu 0.0, typ *boolean* na hodnotu *false* a typ *char* na hodnotu "\u0000". Pro deklaraci proměnných se využije syntaxe (například naplnění proměnné typu *int* na hodnotu 1) int x = 1; [1].

Speciálním případem proměnných jsou proměnné s konstantní hodnotou. Takovéto proměnné jsou v Javě označovány pomocí klíčového slova *final*. Při deklaraci proměnné x jako *final* s hodnotou 10 se použije následující syntaxe:

```
public static void main(String[] args) {
final int x = 10;
}
[Zdroj: Autor]
```

Konstantu x deklarovanou jako *final* již nejde přiřadit jinou hodnotu a to ani stejnou jako měla na počátku. Pokud by došlo k pokusu o změnu hodnoty konstanty x, Eclipse v konzoli vypíše následující chybové hlášení: *The final local variable x cannot be assigned.*

Formátovaný výstup

Formátovaný výstup realizovaný pomocí *System.out.print()* je způsob tisku, kdy se při výstupu číselné proměnné automaticky konvertuje na odpovídající posloupnost čísel. Tato metoda při tisku převede proměnnou, která je deklarována jako její parametr, na řetězec představující hodnotu proměnné a ten následně vytiskne. Například při deklarování proměnné jako *int = 5;* a zavolání metody *System.out.print(i);,* se na výstupu vytiskne řetězec obsahující jeden znak 5 [1].

Pokud nastane situace, při které je nutné doplnit výstup o text pro popsání výstupu, nebo o další proměnnou, je možné toto realizovat následujícím způsobem:

Systém.out.print("Výsledná hodnota proměnné i=" + i + "a hodnota j =" + j*);*

Pro odřádkování lze využít znak \n, ale pohodlnější způsob je využít metodu *Systém.out.println()*.

Formátovaný vstup

Pro vstup dat z klávesnice do proměnné v konzolovém programu je využíváno toho, že Java obsahuje mnoho hotových tříd včetně třídy zajišťující čtení dat z klávesnice. Tato třída nabízí metody pro čtení základních datových typů a řetězců.

V případě, že je nutné využívat třídu *Scanner* je nutné tuto třídu pomocí *import* do programu importovat.

```
import java.util.Scanner; \\ import třídy Scanner
public class Calendar {
            public static void main(String[] args) {…….
[zdroj: Autor]
```

Před prvním čtením vstupu dat z klávesnice je nutné vytvořit instanci třídy *Scanner*. Toho je docíleno pomocí klíčového slova *new*. Na následujícím příkladu je ukázáno, jak by se pomocí importu třídy *Scanner* načítala hodnota typu *integer* do proměnné x a poté by byla vypsána [1].

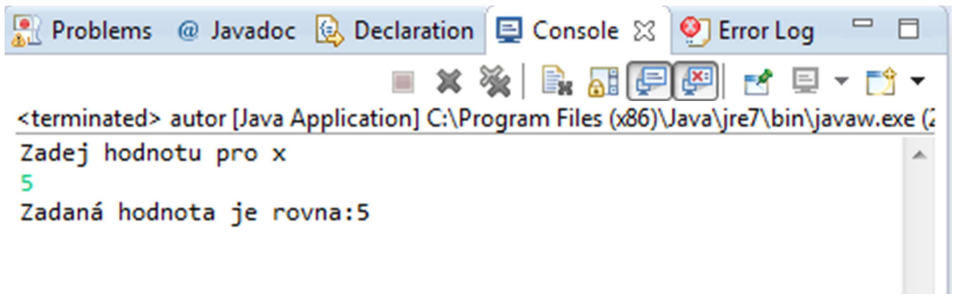
```
import java.util.Scanner;
public class Vstup {

            public static void main(String[] args) {
            System.out.println("Zadej hodnotu pro x");
            Scanner sc = new Scanner(System.in);
            int x = sc.nextInt();
            System.out.print("Zadaná hodnota je rovna:" +x);

      }

}[Zdroj: Autor]
```

Konzolový výpis v Eclipse vypadá následovně:

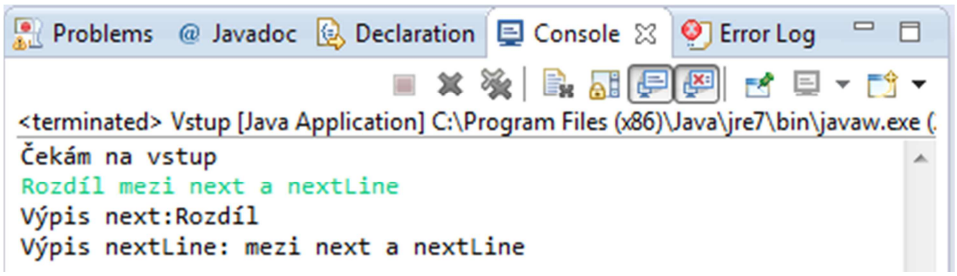


Obrázek 9 Konzolový výpis [Zdroj: Autor]

Pokud je pro běh programu nutné, aby byl z klávesnice čten řetězec znaků, využívá se pro to podobná syntaxe jako při čtení *integeru*. Jedinou změnou je to, že místo *sc.nextInt()* se dosadí *String x = sc.nextLine()*. Při zavolání metody *sc.nextLine()*, je do proměnné načteno vše, co uživatel napíše až do doby, než zadá ukončovací znak. Pokud by byla využita metoda *sc.next(),* do proměnné by se uložilo vše před prvním prázdným znakem (mezerou). Na příkladu na následující straně je tento rozdíl velmi dobře patrný [1, 5].

```
import java.util.Scanner;
public class Vstup {
      public static void main(String[] args) {
            System.out.println("Čekám na vstup");
            Scanner sc = new Scanner(System.in);
            String x = sc.next();
            String y = sc.nextLine();
            System.out.println("Výpis next:" +x);
            System.out.println("Výpis nextLine:" +y);
            }} [Zdroj: Autor]
```

Na výpisu z konzole lze vidět, že po zadání řetězce se do proměnné *x* načte první řetězec až do prázdného znaku. Po zavolání metody *sc.nextLine()* je do proměnné *y* uložen zbylý řetězec na řádce. Rozdíl je tedy mezi tím, že zatímco metoda *cs.next()* načte do proměnné vše ze vstupu do první mezery, tak metoda *sc.nextLine()* načítá řetězec, dokud není ukončen enterem [26].



Obrázek 10 Rozdíl výpisu next a nextLine [Zdroj:Autor]

3.4 XML

Jazyk XML je jedním ze značkovacích jazyků a standardů doporučovaných k ukládání dat. Jazyk je standardizován konsorciem W3C. Tento jazyk vychází z jazyka SGML (Standard Generalized Markup Language) [27].

Značkovací jazyk XML je přenosný, tedy nezávislý na platformě. Přenositelnost je dána tím, že se jedná o prostý text, který ovšem podléhá normám W3C. Tento standard je hojně využíván pro výměnu a sdílení informací a dat. XML pracuje se znakovou sadou ISO 10646. Jedná se o 32bitovou znakovou sadu, která dokáže pojmout veškeré znaky dnes využívaných jazyků. Výhodou je možnost v jednom textu využívat více jazyků najednou.

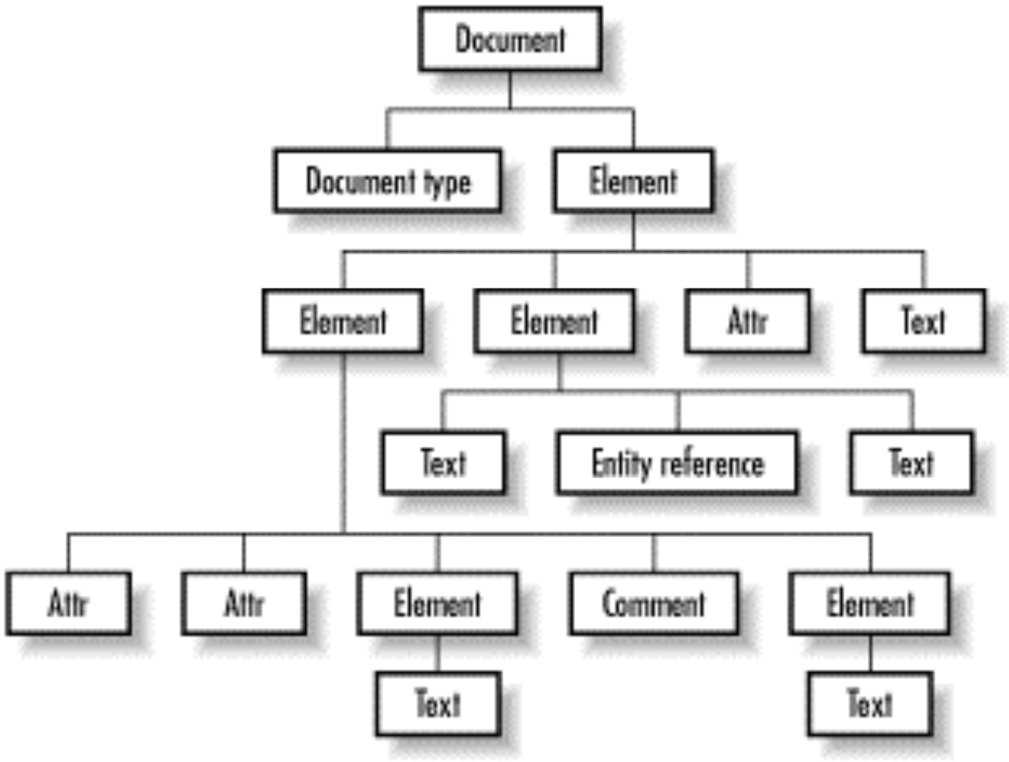
McLaugblin v knize [6] píše, že značkovací jazyk XML využívá pro vyznačení jednotlivých částí textu v dokumentu systému značek. Jedná se například o definování názvu produktu, identifikačního čísla nebo data narození. Tyto značky (elementy) se skládají z počátečního a koncového tagu, jak zobrazuje následující příklad, ve kterém je kořenový element „priklad“ a poté definováno jméno a příjmení autora práce [6].

```
<priklad>
<jmeno>Jan</jmeno>
<prijmeni>Kuník</prijmeni>
</priklad>
```

Názvy tagů je vždy nutné zapisovat mezi znaky ‘<’ a ‘>’. Ukončovací tag má následně před svým názvem ještě znak ‘/’, aby byl snadno odlišitelný od počátečního [28].

3.5 DOM

DOM neboli Document Object Model je aplikační rozhraní, které má stejně jako značkovací jazyk XML své počátky v W3C. Důvodem vzniku byla vzájemná nekompatibilita rozhraní jednotlivých webových prohlížečů. Tato specifikace je jazykově a platformě nezávislá.



Obrázek 11 Stromová struktura [Zdroj: [6]]

Jádrem DOM je stromový model a poskytuje několik rozhraní XML (specifické rozhraní), jako například Element, Document a Text. V typickém dokumentu XML je možné získat strukturu, která je na obrázku 11. Hlavní využití spočívá v jednoduchém přístupu k XML dokumentu, nebo jeho částem a možnost je libovolně upravovat a měnit.

Specifikace W3C Dom je rozdělena do několika DOM úrovní (levels). V současné době se jedná o tři úrovně a čtvrtá se vyvíjí [6].

3.6 Eclipse

V následující kapitole je stručně popsána historie modelovacího nástroje Eclipse a základní informace o tomto nástroji. Vlastní popis prostředí a funkcí je realizován prostřednictvím aplikací v praktické části této práce.

3.6.1 Historie

Společnosti jako Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft a Webgain stály na počátku zrodu nového konsorcia založeného v listopadu 2001. Cílem bylo vytvoření jednotného a univerzálního vývojového

prostředí, které je ovšem založeno na open-source. Nový projekt je znám jako Eclipse. Do konce roku 2003, se počáteční konsorcium rozrostlo na více než 80 členů.

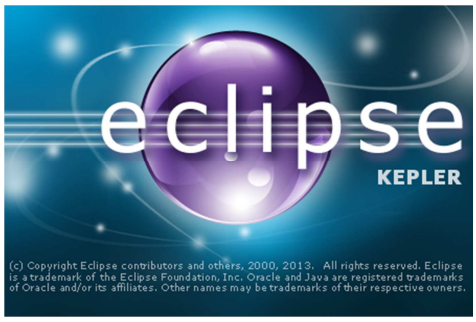
Eclipse je univerzální vývojové prostředí, které je vyvíjeno pod licencí CPL (Common Public Licence) a napsané v jazyce JAVA. Tato licence zaručuje, že je možné do Eclipse přidávat vlastní moduly a dále jej s těmito moduly distribuovat. Tyto moduly ovšem nemusejí mít stejnou licenci [29].

Eclipse je společenství pro jednotlivce a organizace skrz celý softwarový průmysl, kteří si přejí a jsou ochotni spolupracovat na open source softwarech. Jsou to projekty zaměřené na vytváření otevřených vývojových platforem, které se skládají z rozšiřitelných Framework, nástrojů a knihoven pro vytváření, zavedení a správu softwaru během celého jeho životního cyklu.

Eclipse Foundation je nezisková společnost, která zastřešuje projekty Eclipse a pomáhá rozvíjet i open source komunity a ekosystémy doplňkových produktů a služeb. Založena byla v lednu 2004 jako nezávislá nezisková společnost, která měla funkci správce Eclipse komunity [30].

3.6.2 Základní informace

Eclipse je propracovaná a univerzální aplikace, která se velmi často používá jako integrované vývojové prostředí, především pro vývoj a tvorbu programů v programovacím jazyku Java. V porovnání s některými jinými vývojovými prostředími se může zdát, že je Eclipse relativně malý.



Obrázek 12 Eclipse [zdroj: archiv autora]

Ve skutečnosti ale Eclipse nabízí mnohem větší možnosti využití a to hlavně díky svému systému pluginů, neboli přídavných modulů, díky kterým může být velmi dobře použit pro vývoj aplikací v jazycích C, C++, Python, či PHP. Je zde i možnost pro Fortran nebo COBOL. Dále existují pluginy pro tvorbu XML souborů, nebo správu webových aplikací. Využití Eclipse je možné také pro vývoj RCP aplikací (Rich Client Platform), využívaných především ve vnitrofiremních projektech. Tyto webové aplikace jsou určené k tomu, aby poskytovali stejné vlastnosti a funkce, které jsou normálně spojené s desktopovými aplikacemi [5, 31].

Společnost IBM inicializovala v rámci vývojové platformy Eclipse vznik grafického frameworku SWT. Tento framework zaručuje, že se Eclipse bude chovat na všech operačních systémech podobně a to jako nativní aplikace.

4 Vlastní práce

4.1 Popis instalace Eclipse

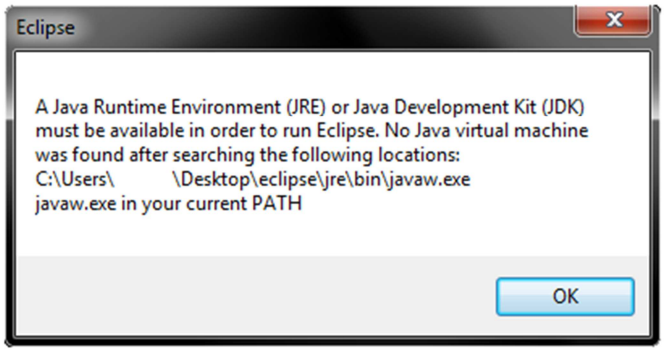
Pokud se uživatel rozhodl začít používat modelovací nástroj Eclipse, nejdříve ho musí nainstalovat na svůj počítač.

Obrázek 13 Stažení Eclipse [zdroj: www.eclipse.org]

Nejjednodušším způsobem je navštívení domovské stránky Eclipse na adrese www.eclipse.org. Zde v pravém horním rohu je umístěn odkaz ke stažení, jak ukazuje obrázek č. 13. Je nutné si dát pozor, jelikož Eclipse je dostupný i pro Linux a Mac OS X, aby byla stažena verze, kterou operační systém bude podporovat.

Po stažení je nutné soubor extrahovat, z důvodu uložení ve formátu .rar. Lze to provést například přes program Total Commander, či jiný program podporující tento typ souborů. Po rozbalení a spuštění instalačního souboru je instalace řízena přehledným průvodcem.

Dalším důležitým krokem je to, že v případě absence JDK, nebo jeho chybného nastavení nahlásí Eclipse chybu a je ukončen. Java Development Kit lze pohodlně stáhnout a nainstalovat z domovské stránky společnosti Oracle. Po úspěšném nainstalování JDK uživateli již nic nebrání spuštění Eclipse.



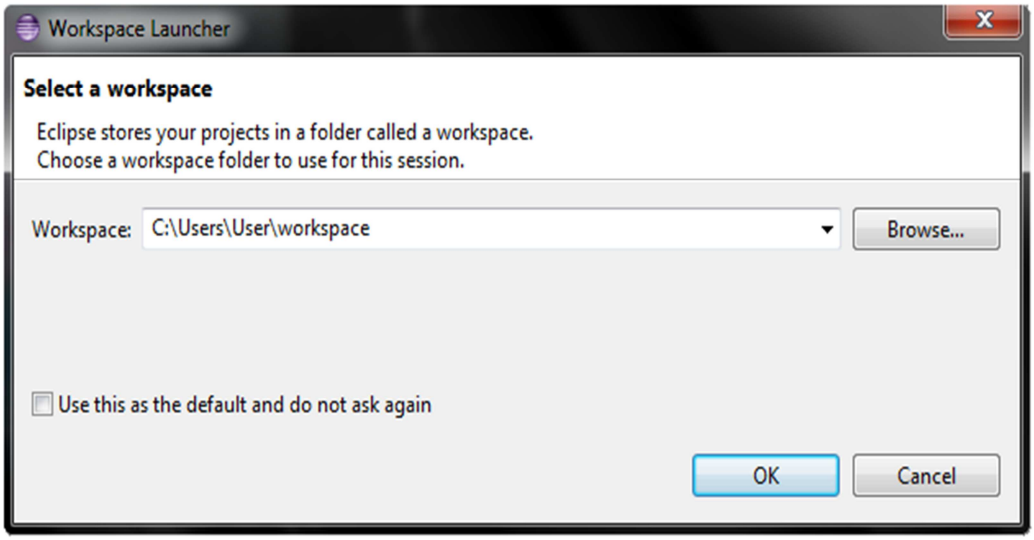
Obrázek 14 Chybové hlášení [zdroj: Autor]

4.2 První spuštění Eclipse

Po úspěšném nainstalování Eclipse otevřeme složku, která byla pomocí zadané cesty při instalaci vytvořena. Nyní již stačí spustit Eclipse pomocí spouštěcího souboru. Tento soubor má následující ikonu  .

Při prvním spuštění programu je nutné nastavit cestu k *workspace* (obrázek č. 15). Jedná se o složku, kterou bude Eclipse využívat jak pracovní. Tato složka může být ponechána tak, jak je nastavena při instalaci a není nutné ji měnit. Pokud uživatel nechce tento krok opakovat při každém spuštění modelovacího prostředí Eclipse, je vhodné zvolit

možnost „*Use this as the default an do not ask again*“. Pokud uživatel zvolí vlastní cestu ke složce, měl by tak učinit na místo, které si zapamatuje a to z důvodu, že se zde ukládají jeho projekty.

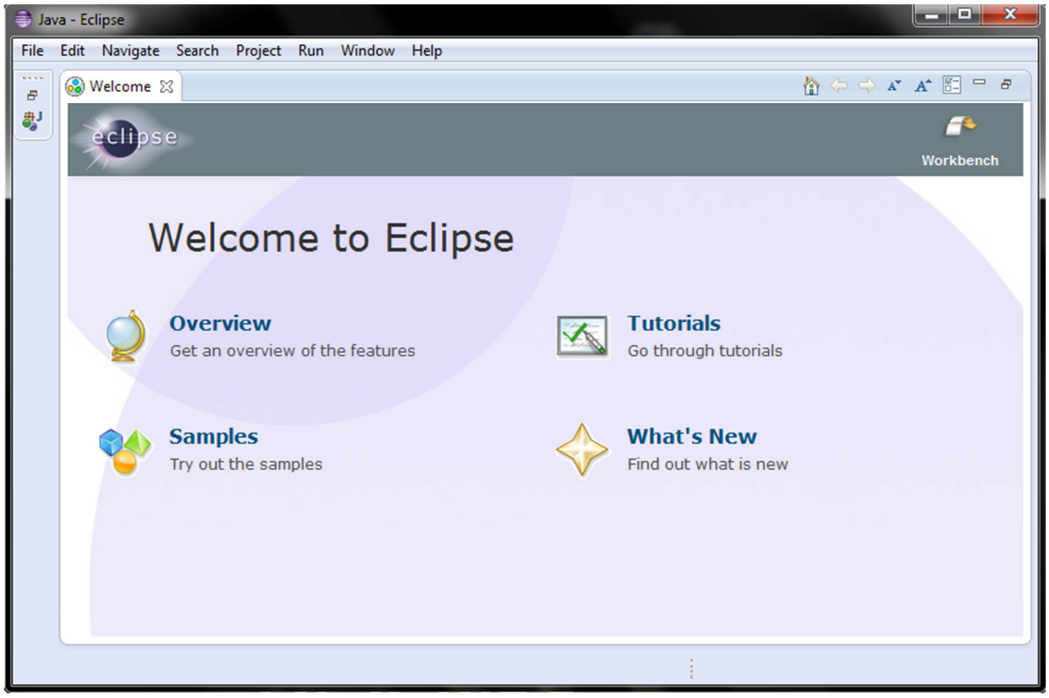


Obrázek 15 Zvolení workspace [zdroj: Autor]

Po nastavení tohoto workspace se uživateli již otevře uvítací okno Eclipse, toto okno je zobrazeno na obrázku číslo 16.

V tomto oknu má uživatel na výběr ze čtyř možností. Za prvé je to *Overview*. Po zvolení této možnosti se uživateli dostane základních informací o modelovacím nástroji. Dále je možnost před další odkazy zjistit jiné důležité informace, například o vývoji Java programů v Eclipse.

Za druhé je to možnost *Tutorial*. Tato možnost nabízí například vytvoření klasického programu „Hello World“. Při zvolení této možnosti je uživateli představeno prostředí Eclipse a to především jak založit nový projekt, jak ho upravovat a na závěr jak ho spustit. Pro uživatele, kteří dříve s tímto modelovacím prostředím nepracovali, je spuštění tutoriálu velmi doporučováno [31].



Obrázek 16 Uvítací obrazovka Eclipse [Zdroj: Autor]

Jako třetí možnost se uživateli zobrazují *Samples.* Při zvolení této možnosti se uživateli naskytne možnost stažení ukázkových programů ze stránek eclipse.org. K tomuto kroku musí mít uživatel přístup k internetu.

Poslední možností je v uvítací obrazovce odkaz *What's New.* Tento odkaz uživateli nabízí možnost zjistit, jaké novinky jsou v poslední aktualizaci Eclipse, či například možnost spojit se s Eclipse komunitou.

Pokud uživatel zavře uvítací obrazovku, zobrazí se mu již modelovací prostředí samotné.

4.3 Popis pracovního prostředí Eclipse

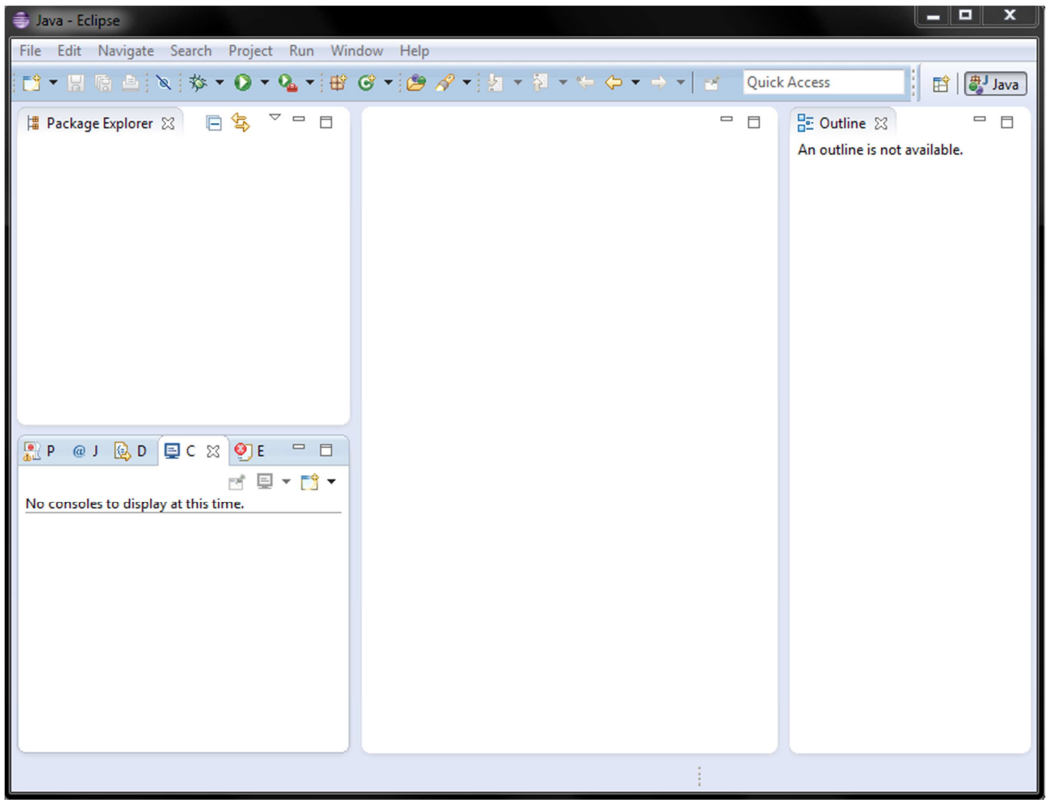
Prostředí je zobrazeno na obrázku číslo 17. Samotné pracovní prostředí je rozděleno do několika jednotlivých částí. Tyto části budou popsány v následující části práce.

4.3.1 Hlavní nabídka

V hlavní nabídce modelovacího prostředí jsou dostupné veškeré funkce a nástroje, které uživateli Eclipse nabízí. Tyto funkce jsou členěny do jednotlivých menu a zobrazují se, pokud je menu aktivováno. V této nabídce nalezne uživatel i menu *help*, ve kterém se

nachází kromě klasické nápovědy programu i odkaz na úvodní stránku, nebo možnost zjistit, zda je uživatelova verze programu Eclipse aktuální.

Pod hlavní nabídkou se nachází nástrojová lišta neboli *toolbar*. V této liště jsou zobrazeny nejpoužívanější funkce a nástroje. Pokud z nějakého důvodu uživateli nevyhovují, lze je samozřejmě upravit. Za jednu z nejdůležitějších lze považovat ikonu *New,* která uživateli umožňuje vytvářet nové projekty, či další potřebné součásti již vytvořeného projektu. Jako další zde nalezneme funkci *Run.* Po spuštění této funkce je prověřeno, zda nejsou v uživatelem napsaném kódu chyby v syntaxi a následně je program spuštěn. Před každým spuštěním musí být napsaný kód uložen. Pokud tomu tak není, objeví se informační okno s dotazem, zda chce uživatel svoji práci uložit.



Obrázek 17 Pracovní prostředí Eclipse [Zdroj: Autor]

4.3.2 Rozložení a popis pracovních oken

Na levé straně se nachází *Package Explorer.* Tato základní část okna Eclipse slouží uživateli pro snadnou správu projektů, knihoven a souborů tím, že zobrazuje uživatelské projekty a jejich přehlednou strukturu. Jsou zde viditelné jednotlivé komponenty nově

vytvořeného projektu, například adresář se zdrojovými kódy a externí archiv obsahující knihovny tvořící standardní API Javy.

V pravé části obrazovky se nachází *Outline.* Tento sloupec zobrazuje veškeré proměnné, třídy a metody, které má uživatel vytvořen v právě otevřeném souboru. Po kliknutí na konkrétní položku se kurzor uživatele přenese na její deklaraci v souboru.

Dále se ve spodní části nachází několik záložek. Jedná se o záložky Problems (na obrázku 17 zobrazeno jako P), Javadoc (na obrázku 17 zobrazeno jako J) a Declaration (na obrázku 17 zobrazeno jako D).

Po spuštění programu je zobrazena další záložka *Console* [na obrázku 17 zobrazeno jako C]. Při prvním spuštění je v této záložce zobrazeno *No consoles to display at this time,* protože v této části pracovního prostředí jsou zobrazeny výstupy z programu a v tomto případě není nic, co by se na konzoly vypsalo.

V záložce *Problems* jsou zobrazeny odkazy na chyby a upozornění. Pokud se takováto chyba vyskytne a zobrazí v této záložce, může uživatel ihned zjistit, kde se v kódu tato chyba nachází.

Záložka *Declaration* zobrazuje uživateli informace o jednotlivých prvcích, kteří jsou v daném kódu deklarovány.

Jako textový výstup programu slouží záložka *Console*. Zde jsou uživateli zobrazeny výstupy z programu.

Editor kódu je v Eclipse umístěn uprostřed pracovního prostředí. Pod pojmem editor si může uživatel představit jakýkoliv běžný textový editor. Programovat Java aplikace je možné například i pomocí běžného poznámkového bloku. V Eclipse je editor kódu, který přehledně zobrazuje napsanou syntaxi a barevně ji rozlišuje podle typů. Například klíčová slova, jako jsou třídy a typy proměnných, jsou zobrazeny pomocí fialové barvy pro lepší přehlednost celkového kódu a snadnější hledání v něm [1, 5].

4.3.3 Perspektivy

Při popisu prostředí Eclipse je nutné zmínit, že je popisována pouze *Java perspektiva*. Perspektivu lze definovat jako souhrn všech prvků na obrazovce. Vývojáři Eclipse tyto

perspektivy zavedli z důvodu, aby nebylo nutné, při každém přechodu mezi například editací kódu k ladění aplikace, měnit rozmístění oken Eclipse. Uživatelské rozmístění oken je právě proto ukládáno ve formě *perspektiv*. Základní nastavení včetně pojmenování je již nastaveno od tvůrců modelovacího prostředí [32].

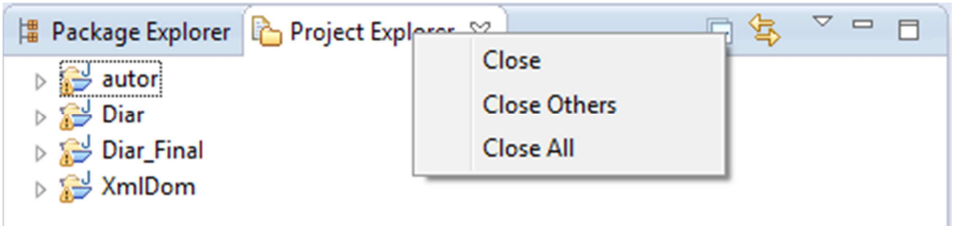
Práce s *perspektivami* je jednoduchá a to i proto, že samotné prostředí Eclipse dokáže při změně činnosti mezi perspektivami přepínat. Uživatel může mezi perspektivami kdykoliv sám volit a to pomocí ikon v pravém horním rohu, či pomocí příkazu na hlavním panelu. Konkrétně se jedná o příkaz *Windows -> Open perspective* a zde si uživatel vybere perspektivu, kterou pro svoji práci bude využívat.

Jako poslední jsou zmíněny pohledy (*View).* Jedná se o jiný typ grafického uživatelského rozhraní (GUI). Jako příklad lze zmínit již popsaný prvek *Package Explorer.* Úkol pohledů je především prezentace aktuálních dat a podporu editorů.

4.3.4 Pohledy

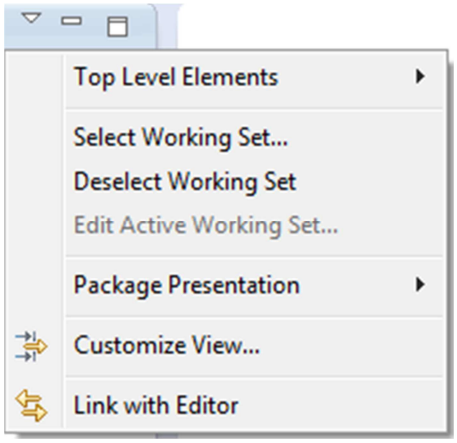
Primární využití pohledů je poskytnout náhled na informace a data, se kterými se v dané chvíli pracuje v *Workbench*. Například se jedná o zobrazení *Bookmarks*, kde se zobrazí všechny záložky v *Workbench* spolu se jmény souborů, se kterými jsou záložky spojeny. Jako další pohled lze uvést *Project Explorer*, který zobrazí veškeré projekty, jejich složky a soubory. Zároveň zobrazuje i vztahy mezi nimi.

Tento pohled disponuje dvěma základními typy menu. Do prvního menu je získán přístup poté, co je využito pravého tlačítka myši v oblasti záložky *View.* Po tomto kroku je zobrazena nabídka na ukončení daného pohledu, v případě, že je otevřeno více pohledů, tak je zobrazena možnost ukončení všech těchto pohledů. Poslední nabízenou možností je ukončení všech pohledu vyjma aktuálně používaného. Tato nabídka je zobrazena na následujícím obrázku.



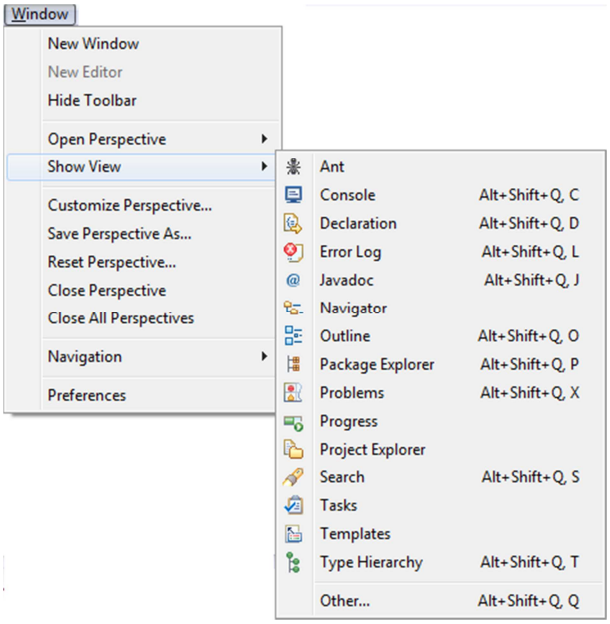
Obrázek 18 Nabídka pohledu [Zdroj: Autor]

Druhé menu, též nazývané jako *view pull-down menu*, je přístupné po aktivaci ikony šipky. Toto menu obsahuje operace aplikovatelné na celý aktuální pohled, ale ne již na jednotlivé části zobrazované v pohledu. Toto menu je zobrazeno na následujícím obrázku [32].



Obrázek 19 View pull-down menu [Zdroj: Autor]

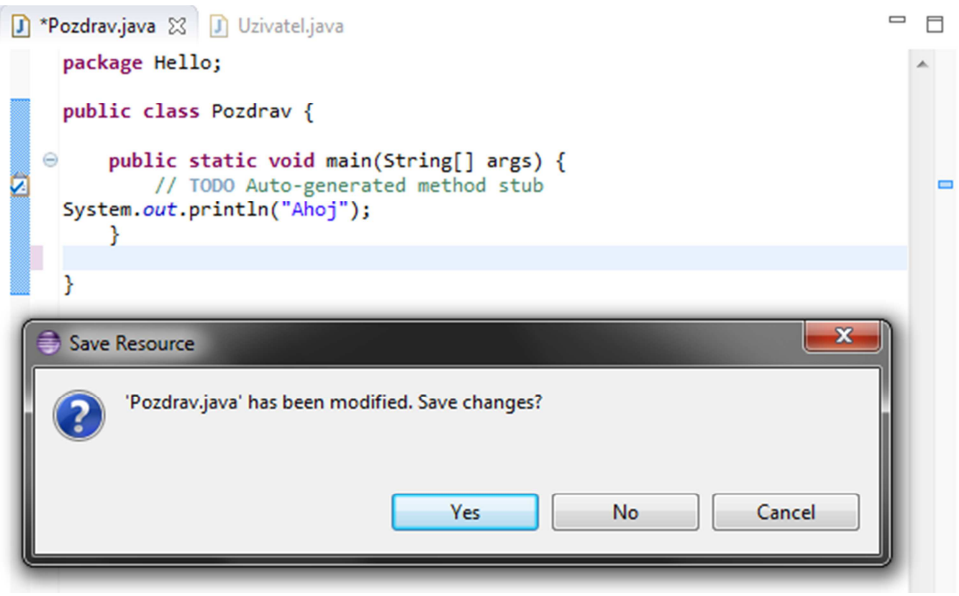
Veškeré pohledy jsou přístupné pomocí záložky *Windows ->Show View*. V této záložce jsou umístěny pohledy, jako například již zmíněný *Package* a *Project Explorer*, dále *Console*, *Debugger*, nebo *Javadoc*. Toto nejsou jediné pohledy, které *Eclipse* nabízí, jak je možné vidět na obrázku číslo 20. V záložce je dále i možnost *Other*, kterou je možné zvolit si dodatečné pohledy. Pomocí *Reset Perspective* je možné vrátit pohled do svého defaultního stavu.



Obrázek 20 View Menu [Zdroj: Autor]

4.3.5 Editor

V návaznosti na tom, jaký soubor je otevřen, Eclipse otevře příslušný editor, který je zobrazen v oblasti určené pro editory. V perspektivě Java se tato oblast nachází uprostřed obrazovky, jak je možné vidět na obrázku 17 *pracovní prostředí Eclipse*. Při otevření souboru k editaci je záložka editoru podle tohoto souboru pojmenován v levém horním rohu. Pokud je se souborem pracováno a proběhly nějaké změny, které nebyly dosud uloženy, je vedle názvu záložky zobrazena hvězdička. Pokud by byla záložka editoru zavřena bez uložení, objeví se okno s dotazem, zda chce programátor změny uložit či nikoliv.



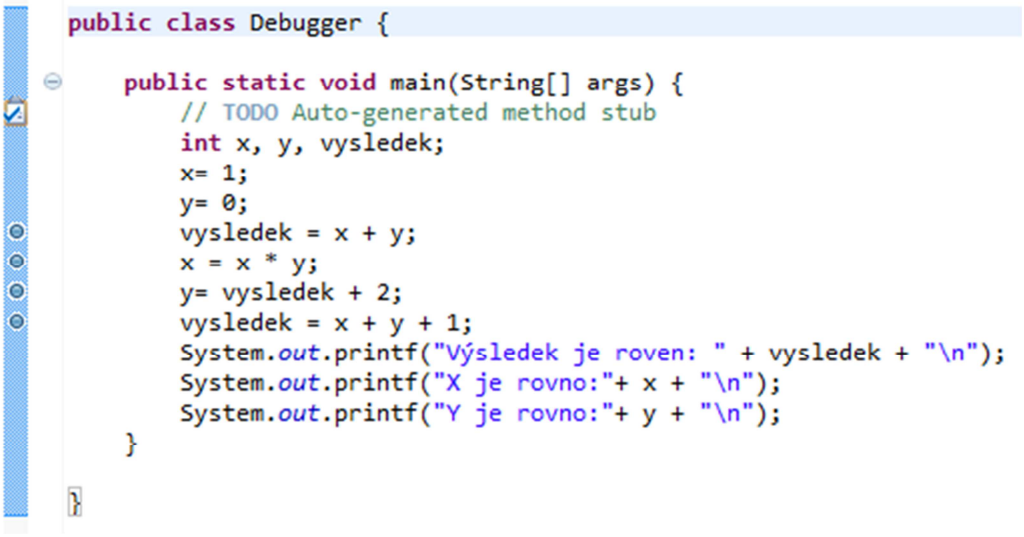
Obrázek 21 Editor kódu [Zdroj: Autor]

4.3.6 Debugger

Java development toolkit (JDT) obsahuje debuger, který umožňuje programátorovi detekovat a diagnostikovat chyby v programech, které jsou spuštěné lokálně nebo vzdáleně. Tento ladící program umožňuje kontrolovat a řídit běh programu pomocí breakpointů, pozastavením běžících programů, krokováním kódu (procházení programem prostřednictvím jednotlivých kroků) a zkoumáním obsahu proměnných.

Před spuštěním debuggeru je nutné nastavit breakpointy, na kterých dojde k pozastavení běhu programu a je umožněno programátorovi bližší zkoumání dané části

kódu. Tyty body jsou na liště zobrazeny pomocí modrých teček a jsou zobrazeny na pro tento případ vytvořeném programu, který sčítá a násobí proměnné a poté jejich obsah vypíše. Na levé straně jsou vidět označené breakpointy.

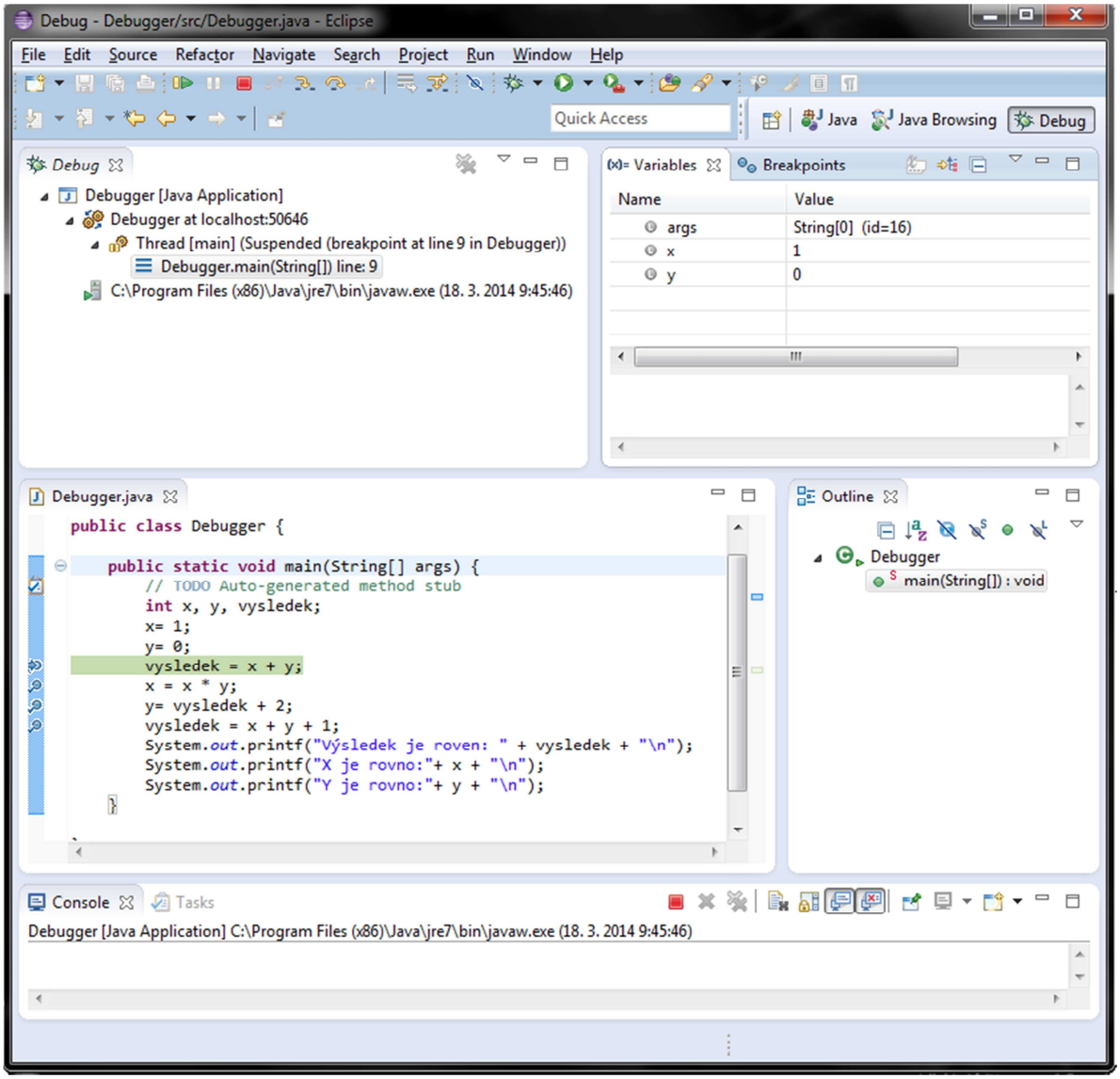


Obrázek 22 Breakpointy [Zdroj: Autor]

Spuštění debuggeru

Při pokusu o spuštění debugeru prostřednictvím ikony  umístěné na hlavní liště, je zobrazeno informační okno s dotazem, zda chce programátor spustit debugger perspektivu. Tato perspektiva slouží pro přehledné krokování a ladění programu. Tato perspektiva je uvedena na obrázku číslo 23 a je zde vidět editor kódu umístěný uprostřed pracovní plochy. Pod tímto editorem je zobrazena konzole. Průběžné hodnoty proměnných jsou přehledně zobrazeny v pravém horním rohu.

Spuštění samotného debuggeru je provedeno pomocí symbolu  . Po každém stisknutí tohoto tlačítka na liště se debugger přesune na další breakpoint a vypíše hodnoty proměnných. V uvedeném příkladu jsou po spuštění debuggeru zobrazeny hodnoty proměnné x a y, konkrétně hodnoty 1 a 0, neboli tak jak byly nastaveny autorem na začátku programu. Po stisknutí tlačítka resume, nebo pomocí F8, je program posunut na další breakpoint a do vypisovaných hodnot se přiřadí i proměnná výsledek s hodnotou 1. Tímto způsobem je možné testovat běh a správnou funkci všech programů, které programátor vytvoří.



Obrázek 23 Debugger perspektiva [Zdroj:Autor]

4.4 Import souborů

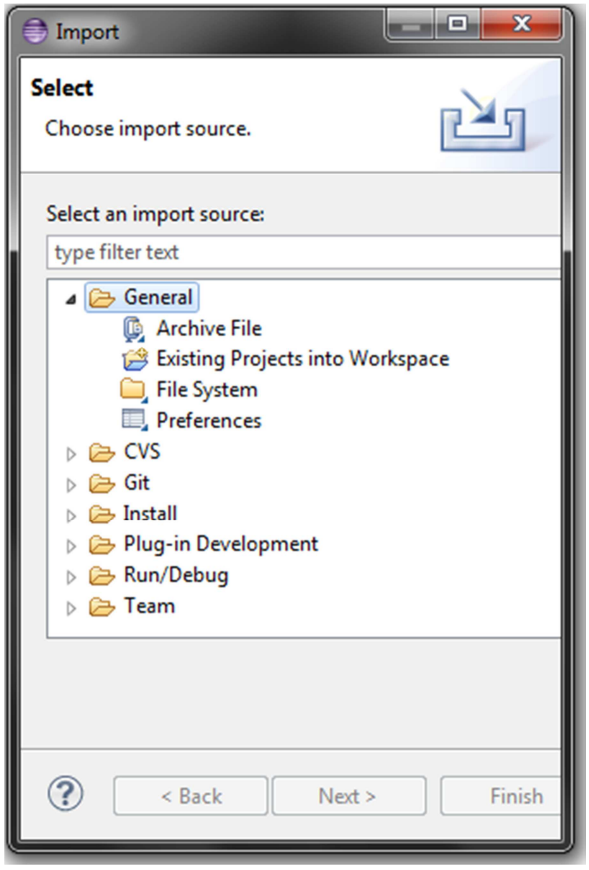
Pokud je zapotřebí, aby proběhl import projektu, který byl vytvořen například někým jiným na jiném počítači, Eclipse nabízí několik možností, jak tento import provést.

Import lze provést pomocí průvodce importem, neboli *Import Wizard*. Ten pomocí několika kroků zajistí, že import souboru bude pro uživatele přehledný a srozumitelný. Postup importu je zobrazen na následujících obrázcích.

Nejdříve je nutné zvolit jaký typ souboru je potřeba naimportovat. Eclipse nabízí v okně *import* řadu možností, jak je vidět na obrázku 24.

Pokud je například potřeba importovat z adresáře souborů, bude zvolena možnost ve skupině *General*. Konkrétně se jedná o možnost *File System.*

Další možností je využití importu již existujícího projektu. V tomto případě se využije možnosti *Existing Projects into Workspace*.

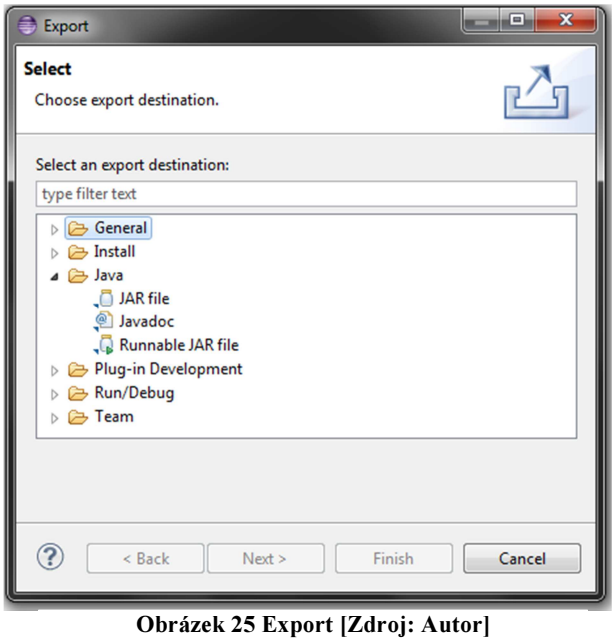


Obrázek 24 Import souboru [Zdroj: Autor]

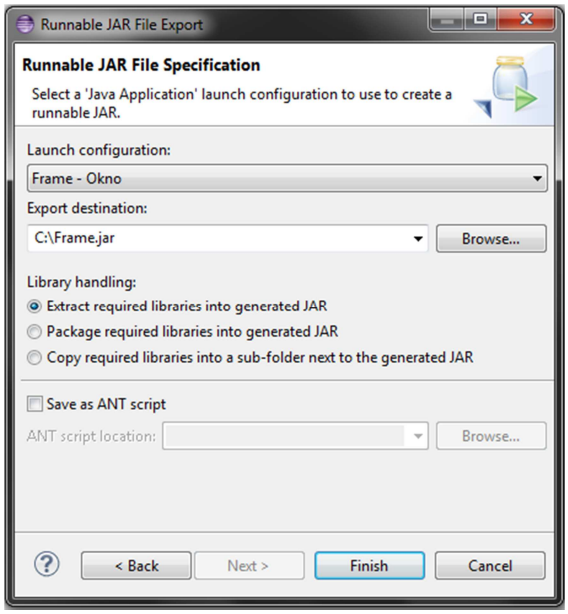
Eclipse nabízí ještě možnost importu, který je proveden při zakládání nového projektu. Při tomto založení je nutné, odznačit možnost *Use default location,* následně pomocí tlačítka *Browse* zvolit cestu k projektu, který je nutné importovat a zvolit tlačítko *Next* pro další konfiguraci, nebo *Finish* pro zahájení importu. Název projektu se automaticky doplní podle názvu, pod kterým je importovaný projekt uložen. Po provedení importu je projekt zobrazen ve *Project Explorer*, tak i v *Package Explorer* [32].

4.5 Export souborů

Export souborů je v Eclipse zajištěn pomocí funkce export dostupné z hlavní nabídky. Na hlavní liště, po zvolení možnosti *File*, je zobrazena nabídka. V této nabídce se nachází možnost *Export*. Na obrázku číslo 25 je zobrazena nabídka, ve které jsou možnosti exportu. Jedná se například o *File System, JAR file, nebo Javadoc*.



Obrázek 25 Export [Zdroj: Autor]



Obrázek 26 Export - destinace [Zdroj: Autor]

Po zvolení daného typu exportu je zobrazena nabídka s destinací, kam má být soubor uložen. Následně je soubor do zvolené destinace vytvořen, jak je vidět na obrázku 26.

Pokud programátor zvolí typ exportu jako *Runnable JAR file,* neboli spustitelný *JAR* soubor, je ihned po exportu možné tento soubor spustit.

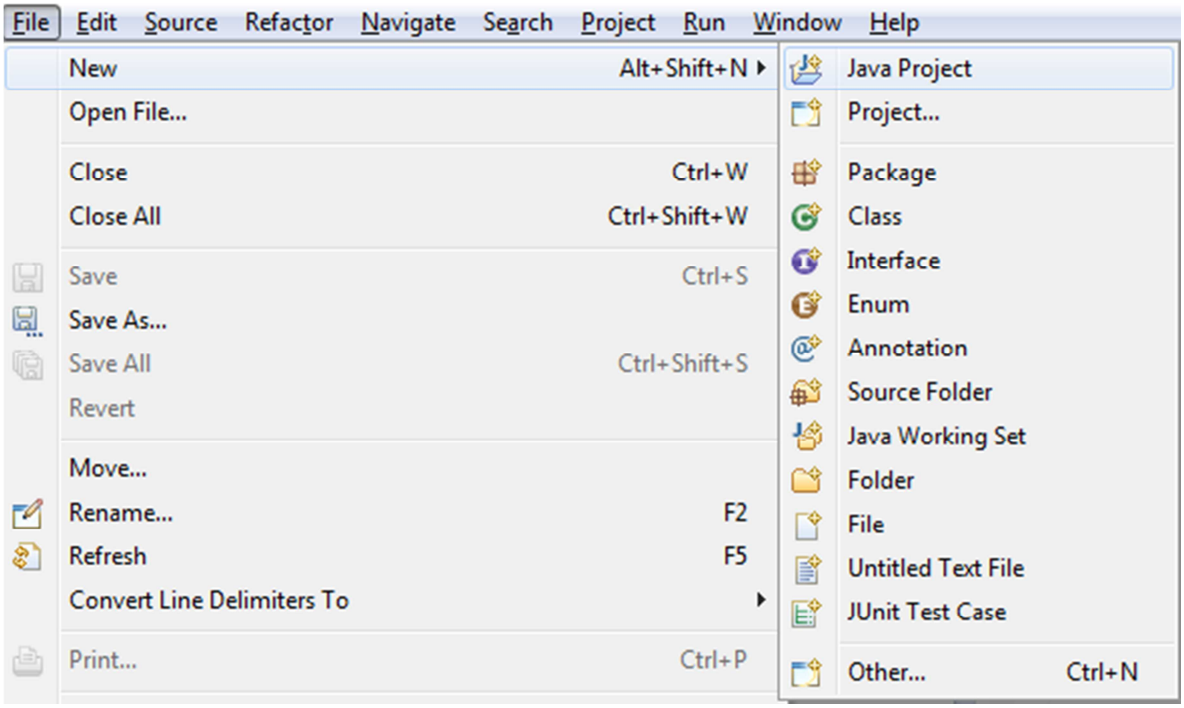
4.6 Založení projektu

Založit nový projekt lze několika způsoby. Záleží na tom, zda je nutné importovat nějaká data, jak bylo již zmíněno výše, nebo se bude jednat o nový projekt.

Pokud je potřeba vytvořit novou aplikaci, jak tomu bylo i v případě dále popisované aplikace, má programátor několik možností jak zajistit vytvoření nového projektu. Možnost vytvořit nový projekt je přístupná přes zvolení možnosti *File* na hlavní liště a následně vybráním *New* (obrázek 27), toto lze povést i pomocí klávesové zkratky. Konkrétně se jedná o kombinaci kláves *Alt* + *Shift* + *N*. Další možnost je kliknutí pravým tlačítkem myši do okna *Package* nebo *Project Explorer*, kde se také zobrazí možnost *New*.

Po zvolení *New* se otevře nabídka, ve které je možnost *Java Project*. Následně jsou zobrazeny možnosti konfigurace nového projektu. Je nutné projekt pojmenovat a určit lokaci na disku, kam se budou data ukládat. Defaultní možností je ta cesta, která je nastavena jako *Workspace* Eclipsu. Eclipse nabízí již teď *Finish*, nebo možnost *Next*.

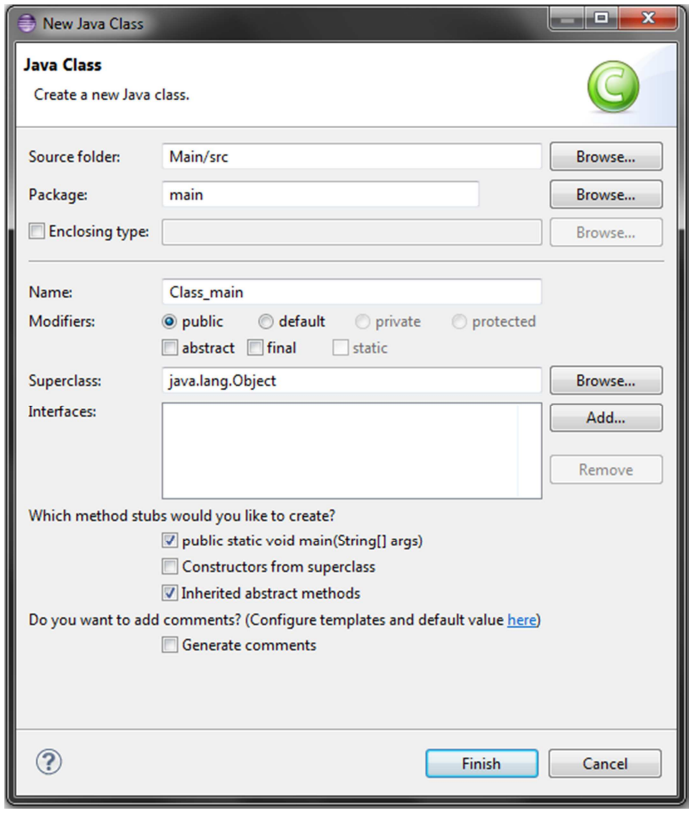
V případě vybrání možnosti *Next* jsou zobrazeny dodatečné možnosti konfigurace. Jedná se například o možnosti přidání knihoven. Po ukončení konfigurace a zvolení *Finish* je vytvořen Java projekt se zvoleným jménem a vlastnostmi.



Obrázek 27 Vytvoření nového projektu [Zdroj: Autor]

Jako další krok je nutné založit nový *package*. Ten slouží k tomu, že když je využíváno objektového programování, tak všechny využité části jsou pod jednou *package.* Celý projekt je pak zobrazen pomocí *Package Explorer*, kde jsou veškeré části projektu přehledně seřazeny.

Po vytvoření *package* je možné již vytvořit konkrétní třídy a jiné součásti, které jsou v daném projektu nezbytné. Pokud je vytvořená nová třída, opět využitím pravého tlačítka myši použitého na využitém *package*, zobrazí se nabídka pro její konfiguraci. V této nabídce lze nadefinovat vlastnosti třídy. Jedná se především o název třídy, její definici jako *public,* nebo například *default* a možnost založit tuto třídu jako *main*. Tyto možnosti jsou vidět na následujícím obrázku, který zobrazuje nabídku možností, které nabízí Eclipse při definování nové třídy.



Obrázek 28 Konfigurace nové třídy [Zdroj: Autor]

Po vytvoření nové třídy, která je typu *public static void main(String[] args)*, Eclipse vytvoří automaticky generovanou definici takovéto třídy. Programátor poté již zapisuje vlastní kód do této třídy. Pokud je potřeba vytvořit další třídu, je vytvořena stejným způsobem jako třída *main*, ale již bez zvolení možnosti *main*.

```
package main;

public class Class_main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}[Zdroj: Autor]
```

Nyní je již možné do této třídy definovat vlastní kód programu. Do vlastní třídy se nadefinují proměnné, které jsou využívány v programu, po tomto nadefinování je již zapsán kód. Vše je vidět na příkladu kódu, který následuje. Tento program si vyžádá od uživatele jméno a to poté vypíše do konzole.

```
package main;

import java.util.Scanner;

public class Class_main {

	public static void main(String[] args) {
	Scanner sc = new Scanner(System.in);
	String name;
	System.out.println("Zadej jméno:");
	name = sc.nextLine();
	System.out.println("Zadané jméno:" + name);
	}
}[Zdroj: Autor]
```

4.7 Vytvoření aplikace s databází

V následující kapitole budou popsány funkce Eclipse při tvorbě diáře, který ukládá záznamy do *xml* souboru. Tento příklad byl zvolen proto, že je na něm dobře vidět práce s daty, ukládání do souboru a načítání ze souboru. Bude také ukázána pomoc, kterou poskytuje Eclipse programátorovy při práci, a tím mu usnadňuje celkovou tvorbu aplikací. Během popisu nebude uváděn celý kód aplikace, ale jen vybrané části. Celý kód je poté zobrazen v příloze.

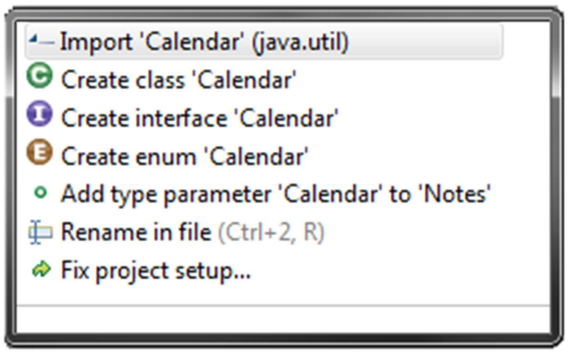
Jako první krok, je nutné opět založit nový projekt, jak je popsáno výše. Po vytvoření a nadefinování třídy byl již definován vlastní kód programu. První třída, která byla vytvořena, byla pojmenována *Notes* a obsahuje záznamy diáře.

Do třídy jsou nadefinované proměnné *dateTime* a *text*. Obě tyto proměnné jsou typu *private*. To z důvodu, aby se k nim nemohlo přistupovat přímo, ale pouze prostřednictvím metod. Tyto metody budou popsány dále.

První proměnná je typu *Calendar*. Tato abstraktní třída v sobě zahrnuje metody pro konverzi mezi konkrétními okamžiky v čase. Dále zahrnuje i sady kalendářních instancí, jako jsou rok, měsíc, den v měsíci, hodina, a tak podobně, a metody pro manipulaci s kalendářem jako s polem. Například se může jednat o získání data příštího týdne [33].

Eclipse při definování proměnné automaticky nabízí možnost importovat *package* java.util, která obsahuje pomocné třídy pro základní programovací struktury, v tomto případě pro práci s časovými formáty. Po provedení importu se nad definicí třídy doplní *import java.util.Calendar*. Celý kód třídy je zobrazen v příloze číslo 4.

U druhé proměnné se jedná o *String*. V této proměnné budou později obsažené konkrétní upomínky, které souvisí s daným datem.



Obrázek 29 Import package Calendar [Zdroj: Autor]

Po importu Eclipse nabízí další možnost automatického doplnění kódu. Jedná se o *gettery a settery* pro obě proměnné. Tyto metody se užívají z důvodu naplnění jednoho ze základních cílů objektově orientovaného programování. Jedná se o skrytí detailů implementace třídy uvnitř této třídy a zároveň zajištění kontroly, které aspekty třídy jsou reprezentovány vnějšímu světu. Následující kód zobrazuje deklaraci zmíněných metod [5].

```
/**
	 * @return the dateTime
	 */
	public Calendar getDateTime() {
		return dateTime;
	}


	/**
	 * @param dateTime the dateTime to set
	 */
	public void setDateTime(Calendar dateTime) {
		this.dateTime = dateTime;
	}
[Zdroj: Autor]
```

Druhá třída, která byla vytvořena, je pojmenována *Databaze*. V této třídě se nachází objekt, do kterého se ukládají zadané záznamy. Tento objekt je typu *ArrayList*. Poté se ve třídě nacházejí metody pro práci s tímto *ArrayListem.* Metody byly pojmenovány dle jejich funkce jako pridejZaznam, najdiZaznamy a vymazZaznamy.

Metoda pridejZaznam je volána prostřednictvím metody v třídě Diar, které bude popsána níže. Po správném zadání je zadáván text konkrétního záznamu. Toto zadání je zapsané do *ArrayListu* a poté je zavolána metoda pro zápis záznamu do souboru.

Metoda najdiZaznamy vyhledává v *ArrayListu* dle zadaného data konkrétní záznamy. Tato metoda vrací nový *ArrayList*, který byl pojmenován *nalezene* a ve kterém se nacházejí vyhledané záznamy.

Metoda vymazZaznamy je využita k tomu, aby se dotázala uživatele, zda opravdu chce smazat celý soubor s poznámkami či nikoliv. Pokud uživatel zadá „Ano“ bude následně smazán celý .xml soubor. Jediným problémem, který se vyskytl při testování aplikace bylo to, že nedošlo ke smazání .xml souboru v případě, že z něj byla načtena poznámka do výpisu na úvodní obrazovce. Zdrojový kód metody je uveden níže. Celý kód třídy je zobrazen v příloze číslo 3.

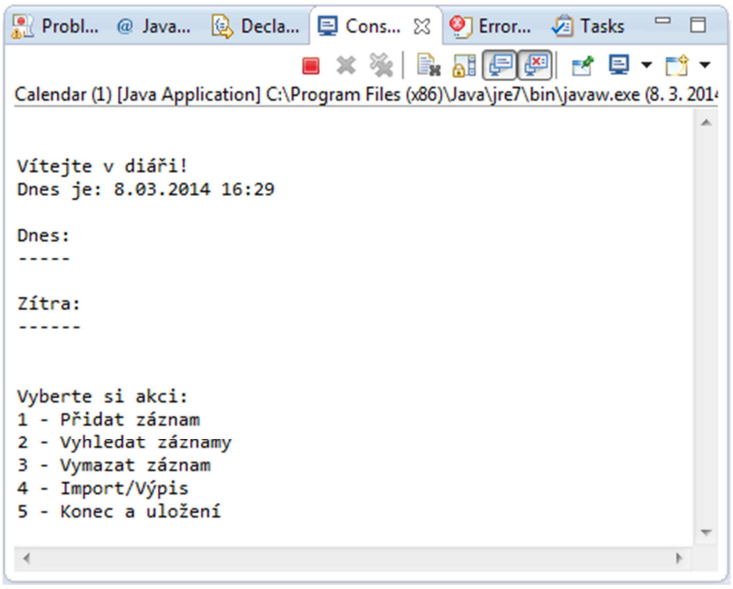
```
      public void vymazZaznamy()
      {

          try{

            File file = new
File("C:\\Users\\Uruma\\Desktop\\Final\\Diar\\diar.xml");

                  if(file.delete())
                        {
                        System.out.println("Soubor diar.xml byl
vymazán!");
                        }
                  else
                        {
                        System.out.println("Soubor diar.xml nebylo možné
smazat.");
                        }}
      catch(Exception e)
                  {
                  e.printStackTrace();
                  }}}
      [Zdroj: Autor]
```

Dalším krokem bylo vytvoření třídy Diar. V této třídě se nacházejí metody pro komunikaci s uživatelem a nedefinovaná metoda *vypisUvodniObrazovku()*, která po zavolání vypisuje aktuální datum. Po tomto výpisu následují zápisy na daný den a na následující den. Dále se vypíší možnosti volby. Tyto volby byly nadefinovány ve třídě *Calendar*. Celý kód je pro názornost zobrazen v příloze číslo 2.



Obrázek 30 Výpis úvodní obrazovky [Zdroj: Autor]

Pro potřeby vyhledávání, přidávání a mazání záznamů, jsou zde nadefinovány dvě možnosti zadávání data a času. Pojmenovány byly jako *formatData* a *formatDataBezCasu.* Vždy je poté definován přesný zápis, který musí být při zadání dodržen. Tato možnost působí jako ochrana před chybovým zadáním a je využita v použitých metodách.

```
public static DateFormat formatData = new SimpleDateFormat("d.MM.yyyy
H:mm");
public static DateFormat formatDataBezCasu = new SimpleDateFormat
("d.MM.yyyy");[Zdroj: Autor]
```

Metoda pridejZaznam je volána ze třídy Calendar při zvolení možnost jedna. Tato metoda jako první krok zavolá metodu *zjistiDatumCas()*, která bude následně popsána. Poté je uživatel vyzván k zadání textu záznamu a tento záznam je uložen do proměnné text, která je spolu s datem zapsána do databáze prostřednictvím metody *pridejZaznam(datumCas, text)*.

Při volání metody *zjistiDatumCas()* je uživatel vyzván k zadání data a hodiny zápisu zprávou `Zadejte datum a čas ve tvaru [1.1.2013 14:00]:`. Tato metoda zavolá metodu *zjistiCalendar* s parametrem *formatData*. V této metodě je potom zadaný datum a čas porovnán s deklarací formatData a v případě, že se neshoduje, je vyvolána výjimka s chybovým výpisem `Nesprávně zadáno, zadejte prosím znovu` a poté je uživatel vyzván k opakování volby.

Jako další metoda ve třídě *Diar* je definována metoda *vyhledejZaznamy()*. Při zavolání této metody je uživatel vyzván následující zprávou, `Zadejte datum ve tvaru [1.1.2013]:`, k zadání data ve formátu den, měsíc a rok. Pro dané datum jsou poté vypsány veškeré záznamy pomocí cyklu *for*.

Poslední třídou, která byla vytvořena, je třída *Calendar*. Tato třída je typu *public static main.* V této třídě se nachází volání metody *vypisUvodniObrazovku()* a textové popsání možností u funkce *switch*. Tato funkce je zakomponována do cyklu *while*, který je ukončen, pokud je volba rovna číslu pět. Jednotlivé volby jsou vidět na obrázku 28 a celý kód třídy je zobrazen v příloze číslo 1.

4.7.1 Zápis do souboru

Jak již bylo zmíněno výše, zápis je proveden do externího souboru, který je typu xml. Pro tento zápis bylo využito třídy *XMLOutputFactory*, která obsahuje *XMLStreamWriter*. Jedná se o *interface*, který určuje, jak bude xml zapsáno. Nezabezpečuje ale již že bude dobře formátován, neboli *well-formed*. Samotný soubor je vytvořen pomocí syntaxe *new FileWriter()*, kde jako parametr je uveden název souboru [22, 34].

Při vytvoření dokumentu bylo nutné zapsat úvodní element, konkrétně pojmenovaný „zaznam“. Dále následuje *for* cyklus, který je vypsán níže. Tento cyklus zapíše počáteční element, poté proměnnou a následně ukončovací element. Po skončení cyklu je soubor uzavřen a poté vyprázdněn buffer pomocí metody *flush()*. Celý kód třídy je zobrazen v příloze číslo 5.

```
      for (Notes n : Databaze.zaznamy)
{
      xsw.writeStartElement("element");
      xsw.writeStartElement("dateTime");

      xsw.writeCharacters(Diar.formatData.format(n.getDateTime().getTime(
      )));
      xsw.writeEndElement();
      xsw.writeStartElement("zaznam");
      xsw.writeCharacters(n.getText());
      xsw.writeEndElement();
      xsw.writeEndElement();
}

         xsw.writeEndElement();
         xsw.writeEndDocument();
         xsw.flush();
      [Zdroj: Autor]
```

4.7.2 Čtení ze souboru

Načtení dat z xml souboru obstarává v aplikaci třída Cteni. V této třídě se využívá balíčku org.w3c.dom, konkrétně třídy Document. Pro ověření správného načtení souboru byl přidán *try-catch* blok s výpisem chybového hlášení. Tento blok je zobrazen níže [5].

```
try
  {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      Document doc = db.parse(new File("diar.xml"));
      ……
   }
catch (Exception e)
      {
       System.err.println("Chyba při čtení souboru: " + e.getMessage());
      }[Zdroj: Autor]
```

Třída DocumentBuilderFactory definuje API (Application Programming Interface) a umožňuje získat metodu *parse,* která načte soubor do objektové struktury.

Pro práci s daty je nutné, aby byl vytvořen kořenový element, z kterého bude zajištěn přesun na další podelementy. Rozhraní *Node*, představuje uzel v dokumentu DOM. Toto rozhraní poskytuje metody, které jsou společné pro všechny uzly. Metoda *getDocumentElement* vrací na *return* objekt *Element*, který představuje kořenový uzel dokumentu. Přes tento kořen je možné vyhledat jiné uzly v dokumentu a najít požadované informace. Metoda *getChildNodes()* získává objekt *NodeList*, který obsahuje všechny podřízené uzly. Samotné čtení bylo zajištěno pomocí cyklu *for*.

Jako další krok, bylo nutné přetypovat uzel na Element pomocí syntaxe `Element element = (Element) uzel;`. Tímto bylo zajištěno získání potřebných metod pro práci se souborem. Jednalo se o metodu `getElementsByTagName()`, která *NodeList* všech prvků se zadaným názvem [35].

Dále bylo využito getteru getTextContent(), který vrátí textový obsah danného uzlu a jeho potomků [5]. Celý kód je zobrazen v příloze číslo 6.

5 Závěr

Cílem této diplomové práce bylo popsat modelovací prostředí Eclipse prostřednictvím programovacího jazyka Java. Tento cíl byl naplněn v teoretické části práce, kde byla popsána historie programovacího jazyka Java, jeho vývoj a úvod do teorie. Dále v této části práce je uvedena historie a základní informace o modelovacím nástroji Eclipse.

Ve vlastní části je popsána instalace a spuštění nástroje Eclipse, na které navazuje popsání pracovního prostředí. Vše je doplněno o ukázku autorem vytvořených aplikací, které zobrazují popisované skutečnosti.

Na závěr práce je popis vytvoření souhrnné aplikace, na které je doplněn popis pracovního prostředí a hlavních funkcí modelovacího prostředí Eclipse.

Eclipse je univerzální vývojové prostředí, které je vyvíjeno pod licencí CPL (Common Public Licence) a napsané v jazyce JAVA. Tato licence zaručuje, že je možné do Eclipse přidávat vlastní moduly a dále jej s těmito moduly distribuovat. Nástroj Eclipse se pro programování v jazyce Java velmi osvědčil. Jedná se o nástroj, který umožňuje programátorovi prostřednictvím mnoha pluginů doplnit funkčnost tak, aby vyhovovala potřebám programátora při tvorbě aplikací.

6 Seznam použitých zdrojů

1. HEROUT, Pavel. *Učebnice jazyka Java.* 2. upr. vyd. České Budějovice: Nakladatelství Kopp, 2001. 349 s. ISBN 80 7232 115 3.
2. PECINOVSKÝ, Rudolf. *Java 5.0: Novinky jazyka a upgrade aplikací.* 1. vyd. Brno: Nakladatelství CP Books, 2005. 153 s. ISBN 80 251 0615 2.
3. MUKHAR, Kevin a kol. *Beginning Java EE 5: From Novice to Professional.* New York: Nakladatelství Springer-Verlag, 2006. 644 s. ISBN 1-59059-470-3.
4. PAVLÍČKOVÁ, J. PAVLÍČEK, L. *Vývoj klient/server aplikací v Javě.* Praha, Oeconomica, 2004. ISBN 8024507919.
5. LOWE, Dough. *Java™ All-in-One Desk Reference For Dummies®.* Indianapolis: Wiley Publishing, Inc., 2005. 862 s. ISBN 978 0 7645 8961 4
6. MCLAUGBLIN, Brett. *Java and XML.* 2. upr. vyd. Vydavatel O'Reilly, 2001. 528 s. ISBN 0 596 00197 5.
7. PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java.* 2. upr. vyd. Praha: Nakladatelství Grada, 2009. 576 s. ISBN 978 80 247 2653 3.
8. SKOUPIL, David. *Úvod do paradigmat programování*. UP Olomouc 2007. 73 s.
9. PAVLÍČKOVÁ J., PAVLÍČEK, L.: *Vývoj klient/server aplikací v Javě*. Praha, Oeconomica 2004. ISBN 8024507919.
10. ČÁPKA, David. *Dědičnost a polymorfismus* [online]. 2013. [cit. 2013-07-13]. Dostupný z WWW: <http://www.devbook.cz/c-sharp-tutorial-dedicnost-a-polymorfismus>.
11. FOJTÍK, Rostislav. *Vývoj objektových aplikací* [online]. Ostrava 2002. [cit. 2013-07-13]. Dostupný z WWW: <http://www1.osu.cz/~fojtik/doc/VOA1.pdf>.
12. KOMISAR, P. *Java history* [online]. 2001. [cit. 2013-08-10]. Dostupný z WWW: < http://www.sentex.net/~pkomisar/Java_History.html>.
13. TIŠNOVSKÝ, Pavel. *Novinky v JDK 7 aneb mírný pokrok v mezích zákona* [online]. 2010. [cit. 2013-08-13]. Dostupný z WWW: <http://www.root.cz/clanky/novinky-v-nbsp-jdk-7-aneb-mirny-pokrok-v-nbsp-mezich-zakona-1/>.
14. DOLEŽAL, Luboš. *Java 8 se opozdí, Oracle se věnuje bezpečnosti* [online]. 2013. [cit. 2013-08-14]. Dostupný z WWW: <http://www.abclinuxu.cz/zpravicky/java-8-se-opozdi-oracle-se-venuje-bezpecnosti>.

15. KOTALA, Z. TOMAN, P. Java [online]. 2001. [cit. 2013-09-09]. Dostupný z WWW: <http://v1.dione.zcu.cz/java/sbornik/4.html>.
16. JAVA.COM, *What is* Java [online]. 2014. [cit. 2014-01-06]. Dostupný z WWW: <http://java.com/en/download/faq/whatis_java.xml>.
17. ORACLE. *Java Card Technology Reference* [online]. [cit. 2014-01-07]. Dostupný z WWW: <http://www.oracle.com/technetwork/java/javacard/documentation/index.html>.
18. ORACLE. *About Java for mobile devices* [online]. [cit. 2014-01-07]. Dostupný z WWW: < http://www.oracle.com/technetwork/java/javame/javamobile/overview/about/index.html >.
19. ORACLE. *Java SE at a Glance* [online] [cit. 2014-01-08]. Dostupný z WWW: <http://www.oracle.com/technetwork/java/javase/overview/index.html>.
20. PINKAS, Jiří, Ing. *Informace: Programovací jazyk Java* [online]. 2007-2014 [cit. 2014-01-8]. Dostupný z WWW: < http://www.java-skoleni.cz/info/java.php >.
21. ORACLE. *The Java tutorials* [online]. [cit. 2014-01-9]. Dostupný z WWW: < http://docs.oracle.com/javase/tutorial/i18n/text/unicode.html>.
22. PELIKÁN, Josef. *Základy jazyka Java* [online]. [cit. 2014-01-10]. Dostupný z WWW: < http://cgg.mff.cuni.cz/~pepca/java/JavaForC++.pdf>.
23. ČÁPKA, David*, Pole v Javě* [online]. [cit. 2014-01-10]. Dostupný z WWW: < http://www.devbook.cz/java-tutorial-pole/>.
24. ČÁPKA, David, *ArrayList v Javě* [online]. [cit. 2014-01-11]. Dostupný z WWW: < http://www.devbook.cz/java-tutorial-list-pridavani-mazani-polozek>.
25. ORACLE. *Class Scanner* [online]. [cit. 2014-01-11]. Dostupný z WWW: < http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>.
26. W3C, *Extensible Markup Language (XML)* [online]. 2013. [cit. 2014-01-11]. Dostupný z WWW: <http://www.w3.org/XML/>.
27. KOSEK, Jiří. *XML pro každého* [online]. 2000. [cit. 2014-01-11]. (PDF) Dostupný z WWW: <http://www.kosek.cz/xml/xmlprokazdeho.pdf>.
28. ZAPLETAL, *Lukáš. Eclipse 2 - IDE na všechno* [online]. [cit. 2013-08-13]. Dostupný z WWW: < http://www.root.cz/clanky/eclipse-2-ide-na-vsechno/>.

29. THE ECLIPSE FOUNDATION, *About the Eclipse Foundation* [online]. [cit. 2013-08-13]. Dostupný z WWW: < http://www.eclipse.org/org/#history>.
30. TIŠŇOVSKÝ, David. *Eclipse – integrované vývojové prostředí pro Javu i další programovací jazyky* [online]. 2012. [cit. 2013-08-14]. Dostupný z WWW: < http://fedora.cz/eclipse-integrovane-vyvojove-prostredi-pro-javu-i-dalsi-programovaci-jazyky/>.
31. ECLIPSE DOCUMENTATION, *Help system* [online]. [cit. 2013-08-20]. Dostupný z WWW: < http://help.eclipse.org >.
32. ORACLE. *Class Calendar* [online]. 2011. [cit. 2013-09-04]. Dostupný z WWW: < http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Calendar.html>.
33. ČÁPKA, David. *Čtení a zápis XML souborů pomocí DOM v Javě* [online]. 2013. [cit. 2013-11-06]. Dostupné z WWW: < http://www.devbook.cz/java-tutorial-xml-dom-cteni >.
34. W3CSCHOOLS, *XML DOM Tutorial* [online]. [cit. 2013-11-06]. Dostupné z WWW: <http://www.w3schools.com/dom/>.

7 Přílohy

V příloze jsou části okomentovaného kódu vytvořené aplikace. Celá aplikace se nachází na přiloženém CD. Tato aplikace je ve formátu .jar.

Příloha 1 Třída Calendar

```
package diary;
import java.util.Scanner;
public class Calendar {
      public static void main(String[] args) {
      Scanner sc = new Scanner(System.in, "Windows-1250");
      Diar diar = new Diar();
      Cteni cteni = new Cteni();
      String volba = "";
                                                            // hlavní
cyklus
             while (!volba.equals("5"))             // Podmínka při
které je cyklus ukončen
             {
                    diar.vypisUvodniObrazovku();         // Volání
metody v třídě Diar
                    System.out.println();
                    System.out.println("Vyberte si akci:");          //
Textový výpis možností
                    System.out.println("1 - Přidat záznam");
                    System.out.println("2 - Vyhledat záznamy");
                    System.out.println("3 - Vymazat záznam");
                    System.out.println("4 - Import/Výpis");
                    System.out.println("5 - Konec a uložení");
                    volba = sc.nextLine();         // Načtení volby
uživatele do proměnné volba
                    System.out.println();
                    // reakce na volbu
                    switch (volba)                 // Na základě
načtené volby je proveden switch
                    {
                           case "1":
                                  diar.pridejZaznam();     //
                                  Volání metody pro přidání
                                  záznamu v třídě Diar
                                  break;
                           case "2":
                                  diar.vyhledejZaznamy();  //
                                   Volání metody pro
                                   vyhledání záznamu v třídě
                                   Diar
                                  break;
                           case "3":
                                  diar.vymazZaznamy();     //
                                  Volání metody pro vymazání
                                  souboru v třídě Diar
                                  break;
                           case "4":
                      cteni.cteni();    // Volání metody pro import dat
                                  a výpis v třídě Diar
                                  break;
```

```
                                case "5":
                            Zapis.zapis();    // Zápis dat do souboru, po
                                              kterém následuje ukončení

System.out.println("Libovolnou klávesou ukončíte program...");
                                                  break;
                                default:
                               System.out.println("Neplatná volba,
                     stiskněte libovolnou klávesu a opakujte volbu.");
                                                  break;  } } } }
```

Příloha 2 Třída Diar

```
      package diary;

      import java.text.DateFormat;
      import java.text.SimpleDateFormat;
      import java.util.ArrayList; // Importy automaticky provedené
nástrojem Eclipse
      import java.util.Calendar;
      import java.util.Scanner;

      public class Diar {
        private Databaze databaze;
        private Scanner sc = new Scanner(System.in, "Windows-1250");
        public static DateFormat formatData = new
SimpleDateFormat("d.MM.yyyy H:mm"); // definování formátu vstupních a
výstupních dat
        public static DateFormat formatDataBezCasu = new
SimpleDateFormat("d.MM.yyyy");
          Cteni cteni = new Cteni();
       public Diar()
         {
           databaze = new Databaze();
         }
      private Calendar zjistiCalendar(DateFormat formatData)
          {
              Calendar datumCas = Calendar.getInstance();
              try
              {
                  datumCas.setTime(formatData.parse(sc.nextLine()));
                  datumCas.setLenient(false);
                  // Používá se k vyvolání výjimky při nesprávném formátu
                  datumCas.getTime();
              }
              catch (Exception e)
              {
                  System.out.println("Nesprávně zadáno, zadejte prosím
znovu");
       // Výpis při chybě
                  return zjistiCalendar(formatData);
              }
              return datumCas;
          }
      private Calendar zjistiDatumCas()
          {
      System.out.println("Zadejte datum a čas ve tvaru [1.1.2013
14:00]:");
      // Vyzvání uživatele k zadání datumu a času
```

```
            return zjistiCalendar(formatData);
        }
        private Calendar zjistiDatum()
        {
            System.out.println("Zadejte datum ve tvaru [1.1.2013]:");
    // Vyzvání uživatele k zadání datumu
            return zjistiCalendar(formatDataBezCasu);
        }




    public void vypisZaznamy(Calendar den)
        {
            ArrayList<Notes> zaznamy = databaze.najdiZaznamy(den,
false);
     // Volání metody najdiZaznamy nad ArrayListem
            for (Notes z : zaznamy)
    // cyklus pro výpis nalezených záznamů
                System.out.println(z);
        }

        public void pridejZaznam()
    // Metoda pro přidání záznamů
        {
            Calendar datumCas = zjistiDatumCas();
    //Volání metody pro zjištění datumu a času
            System.out.println("Zadejte text záznamu:");
            String text = sc.nextLine();
    //Načtení textové poznámky do proměnné text
            databaze.pridejZaznam(datumCas, text);
        }

        public void vyhledejZaznamy()
        {
            // Zadání data uživatelem
            Calendar datumCas = zjistiDatum();
            // Vyhledání záznamů
            ArrayList<Notes> zaznamy = databaze.najdiZaznamy(datumCas,
false);
            // Výpis záznamů
            if (zaznamy.size() > 0)
            {
                System.out.println("Nalezeny tyto záznamy: ");
                for (Notes z : zaznamy)
                    System.out.println(z);
            }
            else
                // Nenalezeno
                System.out.println("Nebyly nalezeny žádné záznamy.");
        }

        public void vymazZaznamy()
        {
            System.out.println("Budou vymazány veškeré záznamy");
          System.out.println("Opravdu chcete vše vymazat?");
          System.out.println("Pro vymazání zadej 'Ano' ");
          Scanner sc = new Scanner(System.in, "Windows-1250");
```

```
            String rozhodnuti = sc.nextLine();
             switch (rozhodnuti)
               {
                        case "Ano":
                                        databaze.vymazZaznamy();
                                        break;
                        case "defaul":

                                        break;
            }
          }


          public void vypisUvodniObrazovku()
          {

              System.out.println();

              System.out.println();
              System.out.println("Vítejte v diáři!");
              Calendar dnes = Calendar.getInstance();
              System.out.printf("Dnes je: %s\n",
formatData.format(dnes.getTime()));
              System.out.println();
              // výpis hlavní obrazovky
              System.out.println("Dnes:\n-----");
              vypisZaznamy(dnes);
              System.out.println();
              System.out.println("Zítra:\n------");
              Calendar zitra = Calendar.getInstance();
              zitra.add(Calendar.DAY_OF_MONTH, 1);
              //zitra.add
              vypisZaznamy(zitra);
              System.out.println();
          }


      }
```

Příloha 3 Třída Databaze

```
	package diary;

	import java.io.File;
	import java.util.ArrayList;
	import java.util.Calendar;

	/** objekt, do kterého se budou ukládat záznamy */
	public class Databaze
	{

		static ArrayList<Notes> zaznamy = new ArrayList<>();

		/**Metoda pro přidání záznamu */
			public void pridejZaznam(Calendar datumCas, String
text)
	{
			  zaznamy.add(new Notes(datumCas, text));
			  Zapis.zapis();
	}
		/** Nalezení záznamů */

		public ArrayList<Notes> najdiZaznamy(Calendar datum, boolean
dleCasu)
			{
				ArrayList<Notes> nalezene = new ArrayList<>();
				for (Notes z : zaznamy)
				{
	if (((dleCasu) && (z.getdateTime().equals(datum))) // dle času a
data
					||
((!dleCasu) && // pouze dle data

(z.getdateTime().get(Calendar.DAY_OF_MONTH) ==
datum.get(Calendar.DAY_OF_MONTH)) && (z.getdateTime().get(Calendar.MONTH)
== datum.get(Calendar.MONTH))) &&
(z.getdateTime().get(Calendar.YEAR) == datum.get(Calendar.YEAR)))
nalezene.add(z);
				}
				return nalezene;
			}

			/**Metoda pro vymazání záznamu */
			public void vymazZaznamy()
			{

try{
```

```
      File file = new
File("C:\\Users\\Uruma\\Desktop\\Final\\Diar\\diar.xml"); //Cesta k
souboru

      if(file.delete()){
                  System.out.println("Soubor diar.xml byl vymazán!");
                  }
      else{
      System.out.println("Soubor diar.xml nebylo možné smazat.");
      }
      }catch(Exception e){
      e.printStackTrace();
                  }}}
```

Příloha 4 Třída Notes

```
package diary;

import java.util.Calendar;

/** Vytvoření třídy Notes, která bude obsahovat záznamy v diáři*/
public class Notes {
private Calendar dateTime;
private String text;

//Tato třída obsahuje gettery a settery vytvořené automaticky pomocí
Eclipse

public Notes (Calendar dateTime, String text)
{
this.dateTime = dateTime;
this.text = text;
}

/**
* @return the datumCas
*/
public Calendar getdateTime() {
return dateTime;
}

/**
* @param datumCas the datumCas to set
* @param dateTime
*/
public void setDatumCas(Calendar datumCas, Calendar dateTime) {
this.dateTime = dateTime;
}

/**
* @return the text
*/
public String getText() {
return text;
}

/**
* @param text the text to set
*/
```

```
public void setText(String text) {
this.text = text;
}

@Override
public String toString()
{
return Diar.formatData.format(dateTime.getTime()) + " " + text;
}

}
```

Příloha 5 Třída Zapis

```
package diary;


import java.io.FileWriter;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamWriter;

public class Zapis {
      Databaze databaze = new Databaze();
 /**
 * @param args the command line arguments
 */
public static void zapis() {
// Zápis uživatelů

XMLStreamWriter xsw = null;
 try{
xsw = XMLOutputFactory.newInstance().createXMLStreamWriter(new
FileWriter("diar.xml"));
      xsw.writeStartDocument();
      xsw.writeStartElement("zaznamy"); //Vytvoření počátečního elementu
      for (Notes n : Databaze.zaznamy)
                  {
                        xsw.writeStartElement("element"); //Vytvoření
počátečního elementu
                   xsw.writeStartElement("dateTime"); //Vytvoření
počátečního elementu

xsw.writeCharacters(Diar.formatData.format(n.getdateTime().getTime()));
                   xsw.writeEndElement();
                   xsw.writeStartElement("zaznam"); //Vytvoření
počátečního elementu
                   xsw.writeCharacters(n.getText());
                   xsw.writeEndElement();
                   xsw.writeEndElement();//Vytvoření koncového elementu
                  }

                  xsw.writeEndElement();
                  xsw.writeEndDocument();
                  xsw.flush();
              }
              catch (Exception e)
              {
```

```
                    System.err.println("Chyba při zápisu: " +
e.getMessage());
                }
                finally
                {
                    try
                    {
                        if (xsw != null)
                        {
                                xsw.close();
                        }
                    }
                    catch (Exception e)
                    {
                            System.err.println("Chyba při uzavírání
souboru: " + e.getMessage());
                    }}}}
```

Příloha 6 Třída Cteni

```
package diary;

import java.io.File;
import java.util.Calendar;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class Cteni {
      public  void cteni() {
      Databaze.zaznamy.clear();
// Vymazání proměnné záznamy, aby nedocházelo k redundanci dat
try        {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(new File("diar.xml")); // Načtení
souboru diar.xml

            Node koren = doc.getDocumentElement();
            koren.normalize(); // normalizace

            NodeList zaznamy = koren.getChildNodes();
      for (int i = 0; i < zaznamy.getLength(); i++) //Cyklus pro
procházení souboru
                {
                    Node uzel = zaznamy.item(i);

if (uzel.getNodeType() == Node.ELEMENT_NODE &&
uzel.getNodeName().equals("element"))
                    {
Element element = (Element) uzel;
String dateTimeText =
element.getElementsByTagName("dateTime").item(0).getTextContent();
```

```
Calendar datumCas = Calendar.getInstance();
datumCas.setTime(Diar.formatData.parse(dateTimeText));
String text =
element.getElementsByTagName("zaznam").item(0).getTextContent();
Notes n = new Notes(datumCas, text);

Databaze.zaznamy.add(new Notes(datumCas, text));
//Naplnění proměnné zaznamy nalezenými daty
System.out.print(n); //Výpis nalezených dat
System.out.print("\n");
      }}}
            catch (Exception e)
              {
                  System.err.println("Chyba při čtení souboru: " +
e.getMessage());
              }}}
```