Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Java aplikace - teorie a praxe

Milan Slabý

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Milan Slabý

Informatika

Název práce

Java aplikace – teorie a praxe

Název anglicky

Java applications – theory and practice

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku vývoje aplikací v jazyce Java. Hlavním cílem práce je charakterizovat základní aspekty, které jazyk Java programátorům nabízí.
Dílčí cíle bakalářské práce jsou:
- analyzovat obecné požadavky na samotný programovací jazyk,
- charakterizovat různé problémy, které s sebou vývoj aplikací přináší a ukázat možná řešení v praxi,
- navrhnout nejschůdnější možné řešení určitých problémů při návrhu a vývoji aplikací.
- zpracovat zkušenosti s vývojem vlastních aplikací.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Dále také na vlastních zkušenostech autora BP získaných při tvorbě aplikací.

Vlastní řešení je realizováno formou návrhů různých metod, kterými lze řešit různé situace a problémy.

Na základě syntézy teoretických poznatků, praktických zkušeností a výsledků vlastního řešení budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

Java aplikace, příkazy, vzory, modely, situace, možnosti, hlediska, problémy, řešení, plánování, návrh

---

Doporučené zdroje informací

HEROUT, P. Java – bohatství knihoven. 3. vyd. České Budějovice: Kopp. 2008. 251 s. ISBN 978-80-7232-368-5.

KISZKA, B. 1001 tipů a triků pro jazyk Java. Brno: Computer Press. 2009. ISBN 978-80-251-2467-3.

PECINOVSKÝ, R. Java 8: Úvod do objektové architektury pro mírně pokročilé. 1. vyd. Praha: Grada. 2014. 655 s. ISBN 978-80-247-4638-8.

SCHILDT, H. Java 7 – Výukový kurz. Brno: Computer Press. 2012. ISBN 97880-251-3748-2.

SCHILDT, H. Mistrovství – Java: Kompletní průvodce vývojáře. 1. vyd. Brno: Computer Press. 2014. ISBN 978-80-251-4145-8.

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

Ing. Čestmír Halbich, CSc.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 28. 10. 2015

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 11. 2015

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 02. 03. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Java aplikace – teorie a praxe" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14. 3. 2016 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Čestmíru Halbichovi, CSc. za jeho odborné vedení, cenné rady a čas, který mi věnoval při psaní mé bakalářské práce, a dále své rodině za podporu, pochopení a trpělivost během celého studia.

Java aplikace – teorie a praxe

Java applications – theory and practice

Souhrn

Bakalářská práce je tematicky zaměřena na problematiku vývoje aplikaci v programovacím jazyce Java a lze ji rozdělit do dvou hlavních částí: teoretickou a praktickou. V začátku teoretické části je popsána historie a vývoj jazyka, jeho stručná charakteristika, popis jednotlivých verzí jazyka a edic Javy. Druhá polovina teoretické části je zaměřena na podrobnější rozbor vybraných syntaktických prvků jazyka Java.

Praktická část je reprezentována desktopovou aplikací umožňující testování členů jednotek dobrovolných hasičů. Celá aplikace je napsána v jazyce Java a jejím hlavním cílem je ukázat možná řešení problémů v konkrétních případech a demonstrovat praktické použití prvků jazyka popsaných v teoretické části.

Summary

Bachelor's thesis is thematically focused on the problem of application development in the Java programming language and can be divided into two main parts: theoretical and practical. In the beginning of the theoretical part is described the history and development of the language, its brief characteristics, description of the individual versions of the language and editions of Java. The second half of the theoretical part is focused on more detailed analysis of selected syntactic elements of the Java language.

The practical part is represented by a desktop application enabling the testing members of fire departments of the voluntary firefighters. The whole application is written in Java and its main aim is to show possible solutions to problems in specific cases and demonstrate the practical application of the elements of the language described in the theoretical part.

Klíčová slova: Java aplikace, třídy, metody, rozhraní, generické typy, kolekce, GUI

Keywords: Java applications, classes, methods, interface, generic types, collections, GUI

Obsah

1. Úvod

Informační a komunikační technologie se v současné době staly součástí našeho každodenního života a obklopují nás takřka na každém kroku. Jejich vývoj je v posledních letech tak rychlý, že pokud si dnes zakoupíte jedno z nejmodernějších zařízení jakéhokoliv druhu (například mobilní telefon), tak během několika měsíců zjistíte, že již vlastníte zastaralý model. Život bez moderních technologií si většina lidí nedokáže ani představit. V mnoha ohledech nám zjednodušuje a zpříjemňuje životy, ale na druhou stranu jsme se na nich staly přímo závislými. I v dnešní době se stále najde početná skupina, zejména u starších generací, odpůrců všech moderních technologií, ať už jsou to chytré mobilní telefony, osobní počítače, notebook, tablety nebo největší světová síť Internet.

S vývojem všech výše zmíněných technologií neodmyslitelně a velmi silně souvisí také vývoj programovacích jazyků, bez jejichž přispění by tato zařízení vůbec nemohla fungovat. Od nižších programovacích jazyků a sekvenčního zpracování programů jsme se postupně dostaly až k vyšším programovacím jazykům, které využívají principů objektově orientovaného programování a umožňují napsání mnohem rozsáhlejších programů, aniž by se tyto programy staly pro vývojáře nepřehlednými. Jedním z velice populárních a vysoce rozšířených, zejména díky rozvoji Internetu a operačního systému Android, objektově orientovaných programovacích jazyků je Java, která se také stala tématem této bakalářské práce.

2. Cíl práce a metodika

2.1. Cíl práce

Tématem bakalářské práce je zaměření na problematiku vývoje aplikací v programovacím jazyce Java. Hlavním cílem práce je charakterizovat vlastnosti jazyka a možnosti, které programátorům Java v současné době nabízí.

Mezi dílčí cíle bakalářské práce lze zařadit následující:

- shrnout historické souvislosti a vývoj jazyka Java
- podrobněji popsat vybrané komplexnější prvky jazyka
- navrhnout a implementovat desktopovou aplikaci pro testování členů jednotek dobrovolných hasičů
- při vývoji aplikace navrhnout nejschůdnější možné řešení konkrétních problémů
- do celé práce zapracovat vlastní zkušenosti s vývojem aplikací

2.2. Metodika

Metodika teoretické i praktické části bakalářské práce je založena na analýze a studiu odborných informačních zdrojů nejen v tištěné, ale i elektronické podobě. Dále zahrnuje vlastní zkušenosti autora práce, které získal při tvorbě vlastních aplikací, při implementaci nových požadavků klienta do již vytvořených aplikací nebo při opravách vzniklých chyb.

Vlastní řešení je realizováno formou desktopové aplikace, která názorně ukazuje praktická řešení různých situací, které mohou nastat u aplikací stejného či podobného typu, tedy u aplikací, které slouží k testování znalostí v různých oblastech. Během vývoje aplikace byly použity principy agilního vývoje software.

Závěry bakalářské práce jsou formulovány na základě syntézy teoretických poznatků, praktických zkušeností a výsledků vlastního řešení.

3. Vznik a historie programovacího jazyka Java

Žádný programovací jazyk nevznikl izolovaně od všeho ostatního, co již bylo vymyšleno, ale jako reakce na nedostatky svých předchůdců nebo jako řešení nově vzniklých problémů souvisejících s inovacemi ve světě počítačů. Java v tomto ohledu nebyla výjimkou.

3.1. Jazyk C

Programovací jazyk C vytvořený v roce 1972 Dennisem Ritchiem ze společnosti *Bell Laboratories* radikálně změnil způsob, s jakým se k programování přistupovalo a s jakým se o programování přemýšlelo. Jedná se o efektivní vysokoúrovňový programovací jazyk, který využívá principy strukturovaného programování. Právě strukturované programování nahrazuje skokové příkazy typu *GOTO*, které výrazně znepřehledňovaly celý kód a ztěžovaly následné hledání chyb a celkové ladění programů. Formálně byl standardizován až v prosinci roku 1989 Americkým národním institutem pro standardizaci (ANSI – *American National Standards Institute*). [4] [7]

Přímými předchůdci jazyka C byl jazyk BCPL, vyvinutý koncem šedesátých let Martinem Richardsem pro psaní kompilátorů, a jazyk B, který byl krátce po uvedení programovacího jazyk BCPL představen Kenem Thompsonem jako jeho rozšíření. Společnost *Bell Laboratories* se rozhodla použít jazyk B pro napsání první verze operačního systému UNIX. Hlavní nevýhodou obou zmíněných jazyků bylo neefektivní využívání paměti. V tehdejší době byla kapacita pamětí poměrně malá a jejich cena byla vysoká. Jazyky BCPL a B alokovali pro své programy stále stejně velkou část bez ohledu na to, zdali program využije všechnu dříve alokovanou paměť. Toto je jeden z hlavních nedostatků, který vedl k hledání nového způsobu řešení a potřebě vytvořit nový programovací jazyk. Jazyk C již umožňuje alokaci proměně velké kapacity paměti. [4] [7]

Jedním z často opomíjených aspektů jazyka C je to, že se jedná o programátorský jazyk. Jedná se o jazyk, který byl navržen, vyvinut a implementován skutečnými, pracujícími programátory a proto se v něm odráží jejich myšlení a přístup k práci. Všechny starší programovací jazyky vznikly na základě rozhodnutí byrokratických výborů nebo jako výsledek akademických aktivit. [7]

Jazyk C znamenal v historii programovacích jazyků obrovský skok kupředu, ale ani on nebyl bezchybný a dokonalý. Jeho hlavním nevýhodou byla jeho složitost. Sice využíval metody strukturovaného programování, ale to stále nevyvrací fakt, že pokud projekt dosáhne určité velikosti, jeho složitost přestane být pro programátora zvládnutelná. Další z jeho nevýhod je skutečnost, že jazyk C postrádá jakousi abstrakci reálného světa. Z dnešního pohledu by bylo možné konstatovat, že mu chybí principy objektově orientovaného programování. [4] [7]

I navzdory všem svým nedostatkům znamenal jazyk C takřka revoluci v programování. Položil základy modernějším programovacím jazykům a můžeme se s ním dodnes setkat.

3.2. Jazyk C++

V roce 1979 vynalezl Bjarne Stroustrup ze společnosti *Bell Laboratories* programovací jazyk C++. Původní název tohoto jazyk byl *C with Classes*, tedy v překladu „C s třídami“. Tento název byl v roce 1983 změněn na C++, pod kterým je známý dodnes. Znaky ++ vyjadřují operátor pro inkrementaci a samo o sobě napovídají, že se nejedná o zcela nový jazyk, ale o rozšíření jazyka C, který byl obohacen o paradigmata objektově orientovaného programování. [4] [7]

Tento jazyk zdokonalil, do té doby vysoce úspěšný a rozšířený, jazyk C a po určitou dobu se zdálo, že konečně existuje „dokonalý“ programovací jazyk, který je možné využít pro tvorbu široké škály programů. S masovým rozšířením Internetu se projevila největší nevýhoda jazyka C++, konkrétně jeho nepřenositelnost na jiné platformy. Programy napsané v jazyce C++ je samozřejmě možné zkompilovat pro různé platformy, nicméně pro kompilaci na konkrétní cíl (platformu, procesor) je nezbytný plný kompilátor jazyka C++. Hlavním problémem je tedy to, že vytvoření kompilátoru je drahé a časově náročné. [4] [7]

3.3. Přínos jazyků C a C++ pro Javu

Java vychází z obou výše zmíněných programovacích jazyků. Od jazyka C si odvodila svojí syntaxi a od jazyka C++ zase převzala svůj objektový model. Ke všemu pak přidává vylepšení a rysy, které odrážejí aktuální stav v oblasti umění programování. [6] [7]

3.4. Vznik Javy

Jazyk Java byl vymyšlen v roce 1991 týmem inženýrů ze společnosti *Sun Microsystems* v rámci tzv. Zeleného projektu. Tento tým tvořili: James Gosling, Patrick Naughton, Chris Warth, Ed Frank a Mike Sheridan. Původní název, převzatý podle stromu rostoucího před Goslingovou kanceláří, jazyka byl Oak. Později se ukázalo, že jazyk s tímto názvem již existuje, a proto se jazyk roku 1995 přejmenoval na Java. První funkční verze tohoto jazyka byla vytvořena za 18 měsíců a veřejné oznámení nového jazyka proběhlo na jaře roku 1995. Na jeho vývoji a dokončování původního prototypu se podílelo mnoho dalších programátorů, ale mezi ty klíčové patřili Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin a tim Lindholm. [4] [6] [7]

Původním impulsem pro vznik Javy byla potřeba na platformě nezávislého jazyka pro tvorbu softwaru, který by byl využíván v zařízeních spotřební elektroniky, jako jsou například mikrovlnné trouby, topinkovače a dálkové ovladače. Druhým a mnohem důležitějším faktorem, který přispěl k rozvoji Javy, byl obrovský rozvoj Internetu, který se objevil v téže době, kdy se Java začala vyvíjet. Autoři Javy si rychle uvědomili, že problémy s přenositelností kódu pro spotřební elektroniku jsou prakticky stejné s těmi, na které naráželi při psaní kódu pro web. Vyřešení původní myšlenky s přenositelností kódu na různé platformy spotřební elektroniky vedla k vyřešení stejného problému v prostředí Internetu, který samozřejmě představoval mnohem širší uplatnění. Právě nezávislost kódu na různých platformách je jednou z klíčových vlastností Javy, která ji pomohla katapultovat do pozice předního programovacího jazyka Internetu. [6] [7]

3.5. Java a C#

Síla a vliv Javy je ve světě programovacích jazyků výrazný. Inovativní vlastnosti Javy, její konstrukce a koncepty položily základy dalších programovacích jazyků. Jazyk C# vytvořený společností *Microsoft* pro prostředí *.NetFramework* je pravděpodobně nejvýznamnějším příkladem. Oba jazyky podporují distribuované programování, sdílejí stejnou základní syntaxi a jejich objektový model je také shodný. I přes určité rozdíly jsou si tyto jazyky velmi podobné a jazyk C# je silný důkaz toho, jak Java změnila způsob přemýšlení nejen o programovacích jazycích, ale i o programování samotném. [6] [7]

4. Základní charakteristika Javy

V následujících kapitolách je uveden souhrn vlastností a „hesel“ spojovaných právě s jazykem Java.

4.1. Bezpečná

Bezpečnost je v prostředí Internetu jednou z nejdůležitějších aspektů každé aplikace. Při stahování programů, zejména apletů, z internetového prostředí se může stát, že se spolu s námi očekávaným kódem stáhne i škodlivý software v podobě počítačového viru nebo jiného nežádaného kódu, který se bude snažit způsobit nějakou škodu nebo získat přístup k datům na disku. Java tento problém vyřešila tím, že všechny programy napsané v tomto jazyce jsou spouštěny virtuálním strojem Javy (*Java Virtual Machine*), který řídí jejich běh. Pokud by se některý program pokoušel provádět své vedlejší aktivity mimo toto prostředí, virtuální stroj by mu v tom zabránil. [6] [7]

4.2. Přenositelná a nezávislá na architektuře

Přenositelnost je patrně klíčovým faktorem, který přispěl k obrovskému rozšíření Javy, a umožnila jí, aby se stala jazykem spojovaným výhradně s prostředím Internetu. Portabilitu aplikací zajišťují dva mechanizmy. Prvním z mechanizmů je tzv. bajtkód, který představuje vysoce optimalizovanou sadu instrukcí spustitelnou v běhovém prostředí Javy. Druhým mechanizmem je již zmínění virtuální stroj Javy, který v původní verzi představoval v podstatě interpret bajtkódu. V současné době virtuální stroj Javy využívá JIT (*Just In Time*) kompilátor bajtkódu pro zvýšení výkonu. Tento kompilátor umožňuje, za běhu programu, podle potřeby kompilovat vybrané části bajtkódu v reálném čase a zbytek kódu je jednoduše interpretován. Celý mechanizmus přenositelnosti kódu funguje tak, že pro každou platformu existuje vlastní virtuální stroj, který je schopen zpracovat bajtkód. Na první pohled by se mohlo zdát, že překlad kódu napsaného v Javě do mezikódu, který následně kompilován, respektive interpretován do nativního kódu, není příliš efektivní a ubírá na výkonu. Důvodem, proč tomu tak není je fakt, který je již výše zmíněn – bajtkód je vysoce optimalizovaný. [6] [7]

4.3. Jednoduchá

Jednoduchost jazyka spočívá v tom, že je poměrně lehké naučit se jeho syntaxi. Zejména pokud programátoři již znají nějaký jiný objektově orientovaný jazyk, tak nebudou mít větší problém přejít právě k Javě. Jak již bylo popsáno v kapitole o historii Javy, syntaxe je odvozena od jazyka C, který byl a stále je rozšířeným jazykem, tudíž jeho syntaxe je obecně velmi známá. [6] [7]

4.4. Objektově orientovaná

Java patří mezi objektově orientované programovací jazyky využívající tři základní paradigmata objektově orientovaného programování, kterými jsou:

- **dědičnost** – objekt získá vlastnosti druhého objektu.
- **zapouzdření** – data a metody, které spolu souvisí, jsou uloženy (zapouzdřeny) v rámci jedné třídy.
- **polymorfismus** - dvě metody mohou mít stejný název, ale ne signaturu.

Tento jazyk nelze označit za čistě objektový, protože primitivní datové typy nejsou objekty, ale představujíc jednoduché hodnoty, které mají explicitně daný rozsah možných hodnot a jejich chování je jasně dané z matematického hlediska. Důvodem je zvýšení výkonu.

[6] [7]

4.5. Robustní

Silné typování jazyka, který provádí kontrolu kódu programu již v době kompilace a následně pak další kontroly za běhu programu prakticky zamezuje napsání takového programu, jehož chyby by byly těžko vysledovatelné. Další důležitou vlastností Javy je správa paměti. O alokaci paměti se programátor nemusí starat, protože to za něj zajistí samotný jazyk a dealokace paměti probíhá automaticky a provádí ji tzv. *garbage collection*, který vyhledává nepotřebné, respektive nepoužívané objekty, které ruší a tím uvolní dříve alokovanou paměť. [6] [7]

5. Verze Javy

Od svého vzniku až po současnost vyšlo mnoho verzí Javy, z nichž každá přinesla nějaké vylepšení jazyka nebo přidala zcela novou funkcionalitu umožňující řešení nových i stávajících problémů nebo podporu nově vzniklých technologií. Nové verze současně s inovacemi také označily některé vlastnosti z původních verzí za zastaralé a nadále nepodporované.

5.1. Verze 1.0

První oficiálně vydaná verze jazyka z roku 1995. Sada pro vývojáře (*JDK*[1] *– Java Development Kit*) s označením JDK 1.0 byla dostupná ke stažení 23. ledna 1996 a obsahovala následující balíčky:

- **java.lang** – obsahuje základní třídy Javy.
- **java.io** – poskytuje třídy o rozhraní pro řízení vstupně výstupních proudů pro čtení a zápis dat do souborů nebo jiných zdrojů.
- **java.util** – obsahuje různorodé třídy pro práci se základními strukturami, časem, datem, generátorem náhodných čísel a mnoha dalšími.
- **java.net** – balíček poskytující podporu pro práci v síti.
- **java.awt** – obsahuje třídy pro vytváření grafických uživatelských rozhraní.
- **java.awt.image** – balíček poskytující funkcionality pro práci s obrázky.
- **java.awt.peer** – propojuje AWT[2] komponenty s jejich implementací na konkrétních platformách
- **java.applet** – balíček obsahující třídy pro tvorbu appletů[3].

[8] [16] [17] [19]

[1] JDK = *Java Development Kit* - označení sady pro vývojáře, která obsahuje všechny potřebné knihovny a nástroje pro vývoj aplikací v Javě.

[2] AWT = *Abstract Window Toolkit* - jedná se o balíček obsahující různé komponenty pro vytváření grafického uživatelského rozhraní, jako jsou například okna, tlačítka, textová pole, seznamy a další.

[3] Applet je speciální druh programu, který se automaticky spouští přímo ve webovém prohlížeči podporujícím Javu.

5.2. Verze 1.1

Třináct měsíců po vydání první verze byla uvolněna verze 1.1, která byla 19. února 1997 volně stažitelná ze stránek společnosti *Sun Microsystems*. Mezi zásadní nové prvky této verze lze zařadit následující:

- JDBC[4] – umožňuje propojení jazyka Java s širokou škálou databázových systémů
- Vnitřní třídy – jedna třída v sobě může obsahovat jednu či více dalších tříd
- *Java Beans* – softwarová komponenta, která může být opakovaně použitelná v různých prostředích.
- RMI[5] – vzdálené volání metod
- Reflexe – schopnost softwaru analyzovat sebe sama. Umožňuje zjistit, jaké konstruktory, metody a pole konkrétní třída podporuje v době běhu.

[6] [7] [16] [17] [19]

5.3. Verze 1.2

8. prosince roku 1998 vyšla nová verze Java 2, která je považována za „druhou generaci“. Poněkud zvláštní označení čísla verze 1.2 je zaviněno tím, že se čísla verzí jazyka původně vztahovala k interním číslům verzí knihoven. Společnost *Sun* později celý produkt označila jako J2SE = *Java 2 Standard Edition*, přičemž číslování verzí pokračuje od čísla 1.2. Zásadní změny této verze:

- přidání frameworku *Swing*
- přidání frameworku *Collections*
- prohlášení metod *suspend(), resume()* a *stop()* ze třídy *Thread* za zastaralé a nadále nepodporované

[6] [7] [16] [17] [19]

[4] JDBC = *Java Database Connectivity* - poskytuje rozhraní pro práci s různými druhy databázových systémů.

[5] RMI = *Remote Method Invocation* - Javový objekt spuštění na jednom počítači může volat metody jiného javového objektu spouštěném na jiném počítači.

5.4. Verze 1.3

Jedná se převážně o rozšíření stávající funkcionality z předchozí verze. Pro vývojáře byla uvolněna 8. května roku 2000 a zdrojové kódy napsané ve verzi 1.2 a 1.3 jsou vzájemně kompatibilní. Mezi nejvýznamnější novinky této verze patří podpora pro práci se zvuky a indexování JAR[6] souborů, které umožnilo optimalizovat vyhledávání *class* souborů při jejich načítání pro síťové aplikace, zejména pro applety.

[6] [7] [10] [17] [19]

5.5. Verze 1.4

Necelý rok a půl (konkrétně 6. února 2002) po uvolnění předchozí verze se objevila verze 1.4, která opětovně rozšířila funkcionalitu předchozích verzí a aktualizovala některé stávající prvky. Navzdory všem svým rozšířením a změnám si verze 1.4 zachovala téměř stoprocentní kompatibilitu s předchozími verzemi. Nové prvky této verze:

- přidání klíčového slova ***assert***[7]
- zřetězování výjimek
- I/O[8] subsystém založený na kanálech (NIO[9])
- zpracování XML[10]
- podpora IPv6[11]
- regulární výrazy
- rozšíření *Collections* frameworku
- rozšíření tříd pro práci se sítí

[6] [7] [17] [19]

---

[6] JAR = *Java Archive* – je formát souboru/archivu založeném na formátu ZIP, který umožňuje seskupit všechny třídy a jiné zdroje (obrázky, zvuky, balíčky) do jednoho souboru.

[7] *assert* – klíčové slovo jazyka Java, které se používá zejména při testování.

[8] I/O = *Input/Output* – označení pro vstupně výstupní operace

[9] NIO = *New Input/Output* – označení pro nový I/O založených na kanálech

[10] XML = *Extensible Markup Language* – značkovací jazyk pro tvorbu strukturovaných dokumentů nezávislých na formátování

[11] IPv6 = *Internet Protocol version 6* – internetový protokol verze 6

5.6. J2SE 5

Java 2 *Standard Edition* verze 5 je označována jako druhá revoluce v Javě. Tato verze uvolněná 30. září 2004 přinesla zásadní rozšíření a nové dimenze jazyka. O zásadnosti těchto změn svědčí i to, že se společnost *Sun Microsystems* rozhodla označit celou verzi číslem 5, aby ještě více zdůraznila významnost a velikost nových prvků. Vzhledem k zachování konzistence v číslování verzí se jako interní číslo verze používá 1.5, které se někdy označuje jako vývojářské číslo verze. Produktové číslo verze pro své označení využívá číslici 5. Nové vlastnosti a funkce této verze:

- generické typy
- anotace
- *autoboxing*[12] a *autounboxing*[13]
- výčtové typy
- rozšíření cyklu *for* pro procházení kolekcí
- *varargs*[14] – argumenty s proměnnou délkou
- statický import
- formátovaný vstup a výstup
- nové podpůrné třídy pro paralelismus

Všechny výše uvedené změny, rozšíření a nové funkcionality představují zcela zásadní vylepšení celého jazyka. Prvky jako například generické typy, rozšíření cyklus *for* pro procházení kolekcí nebo argumenty s proměnnou délkou přidaly do Javy nové prvky její syntaxe. K úpravě sémantické stránky celého jazyka přispěl *autoboxing* a *autounboxing*. Zcela novou dimenzi jazyka představují anotace, které umožňují přidávání dodatečných informací do zdrojového kódu programu.

[6] [7] [17] [19]

[12] *autoboxing* – primitivní datové typy jsou automaticky zapouzdřeny do příslušných obalových typů

[13] *autounboxing* – hodnota v obalovém datovém typu je automaticky načítána do příslušného primitivního datového typu

[14] *varargs* = *Variable Arguments* – zkratka označující proměnný počet argumentů

5.7. Java SE 6

Vychází z předchozí verze J2SE 5 a sama o sobě nepřinesla žádné nové prvky a funkce. Přináší vylepšení stávající funkcionality. Za zmínku stojí rozšíření knihovny API a vylepšení samotného běhového prostředí. Společnost *Sun Microsystems* se v této verzi opětovně rozhodla změnit její název a to vynecháním číslice 2 z názvu platformy. Interní číslo verze neslo označení 1.6, sada pro vývojáře JDK 6 a produktové číslo verze 6. Byla uvolněna 11. prosince 2006. [6] [7]

5.8. Java SE 7

První verze Javy vydaná společností *Oracle*, která v roce 2010 koupila společnost *Sun Microsystems*. Vydání této verze proběhlo 28. července 2011 a přineslo mnoho nových vlastností a funkcí celého jazyka. Výčet těch nejdůležitějších je uveden níže:

- příkaz *switch* může být řízen objektem typu *String*
- binární celočíselné literály
- použití podtržítka v numerických literálech
- příkaz *try-with-resources* – umožňuje automatickou správu prostředků, například automatické uzavírání proudů nebo souborů v době, kdy již nejsou potřené
- operátor diamant
- *multi-catch* – umožňuje zachytit dvě a více výjimek jediným příkazem *catch*
- zásadní rozšíření frameworku *NIO* – často se používá označení *NIO.2*
- přidání frameworku *Fork/Join* – zajišťuje podporu pro paralelní programování

Všechny uvedené změny byly vyvinuty v projektu označovaného jako *Project Coin*. Všechna výše zmíněná rozšíření byla v rámci projektu označována jako „malé“ změny, ale jejich dopad rozhodně za malý považovat nelze.

[6] [7] [17] [19]

5.9. Java SE 8

V době psaní této bakalářské práce se jedná o poslední verzi, která byla uvolněna 18. března roku 2014. Osobně bych tuto verzi označil také za revoluční, protože přidala některé prvky funkcionálního programování, rozšířila stávající knihovny, zvýšila bezpečnost a rozšířila se na nové typy zařízení, jako jsou například různé sensory umístěné v hodinkách nebo čipech na zvířatech. Nejzásadnější změny jsou podrobněji popsány níže. [17]

5.9.1. Lambda výrazy

Nový prvek Javy umožňující funkcionální programování za pomoci Lambda výrazů. Lambda výrazy představují bezejmenné funkce, které lze použít i jako parametry metod, umožňující filtrovat vstupní hodnoty, na jejichž základě je posléze určena hodnota výstupu. Dále tyto výrazy zlepšili práci s kolekcemi, které nyní lze velmi snadno iterovat, filtrovat a získávat z nich potřebná data. [14] [20] [23]

5.9.2. Internet věcí

Internet věcí je společností *Oracle* označován za jednu z největších novinek Javy 8. Prakticky tento termín označuje vše, co je možné připojit k internetu nebo jiné interní síti. Jedná se především o různé senzory umístěné v chytrých hodinkách nebo náramcích snímající tep, různá čidla v chytrých domácnostech nebo automobilech. V neposlední řadě také aplikace, které přijímají data z výše uvedených zdrojů a vyhodnocují je. [20]

5.9.3. Nashorn

Jedná se o projekt představující nový JavaScriptový *engine* běžící přímo v běhovém prostředí Javy (JVM[15]). Tento *engine* vývojářům umožňuje využívat prvky Javy i JavaScriptu v jedné aplikaci bez ztráty výkonu. To je způsobeno tím, že se kód napsaný v JavaScriptu překompiluje do bajt kódu, který je následně vykonán interpretem.

[20] [23]

[15] JVM = *Java Virtual Machine* – virtuální stroj Javy umožňující běh aplikací

5.9.4. Nové balíčky pro datum a čas

Nové balíčky obsahující mnoho tříd pro práci s datem a časem včetně časových zón. Tyto balíčky podporují reprezentaci pouze času bez data nebo naopak pouze data bez času. Další výhodou je možnost podvržení času komponentám, což je užitečné po testovací účely. Základním balíčkem je *java.time*, který obsahuje několik podbalíčků. Jejich výčet je uveden zde:

- *java.time.chrono* – obsahuje třídy pro systém kalendářů jiných, než definuje ISO-8601
- *java.time.format* – třídy pro formátování a převod data a času
- *java.time.temporal* – primárně určená pro implementaci knihoven a frameworků, umožňuje provádění vnitřních operací s datem a časem
- *java.time.zone* – třídy podporující časové zóny

[11] [20] [23]

5.9.5. Stream API

Souvisí především s Lambda výrazy a kolekce, kterým nově přibyla metoda *stream()*, která vrací funkcionální pohled. Na tento pohled potom můžeme aplikovat různé operace, například v podobě různých filtrů. Důležitým faktem je to, že změny ve vráceném pohledu nemají žádný vliv na kolekci. [20] [23]

5.9.6. Zabezpečení

Mezi novinky v zabezpečení, které se v Javě 8 objevily, můžeme zařadit například podporu TLS[16]. Jedná se o protokoly zabezpečující komunikaci na Internetu zejména proti odposlouchávání nebo falšování zpráv pomocí autentizace pro koncový bod.

Další novinkou je možnost šifrování dat za podpory *NSA Suite B Cryptographic Algorithms*. Jedná se o balíček velmi silných algoritmů sloužících k šifrování. Některé šifrovací algoritmy z toho balíčku používají 128 nebo 256 bitů dlouhé klíče. Dále balíček obsahuje algoritmus pro podporu digitálního podpisu ECDSA[17]. [20] [23]

[16] TLS = *Transport Layer Security* – protokoly pro zabezpečenou komunikaci na Internetu

[17] ECDSA = *Elliptic Curve Digital Signature Algorithm* – algoritmus pro podporu digitálního podpisu

6. Edice Javy

Společnost *Oracle* v současné době nabízí tři základní edice Javy, z nichž každá je předurčena pro různé typy aplikací a rozdílné potřeby skupin uživatelů. Součástí každé edice je virtuální stroj Javy (JVM - Java Virtual Machine) a API[18]. Stručný popis všech edic je uveden níže. [24]

6.1. Java ME

Java *Micro Edition*, označovaná zkratkou Java ME, poskytuje robustní a flexibilní prostředí pro běh aplikací na mobilních a tzv. *embedded* zařízeních, souhrnně označovaných jako Internet věcí. Obsahuje přizpůsobivá uživatelská rozhraní, silné zabezpečení, které je vestavěné do síťových protokolů, a podporu pro síťové aplikace a *offline* aplikace, které mohou být dynamicky stahovány. Aplikace založené na této platformě jsou přenositelné napříč mnoha zařízeními. Lze ji označit jako podmnožinu Java SE. [13]

6.2. Java SE

Java SE (*Standard Edition*) je platforma umožňující vytvářet desktopové aplikace určené pro stolní počítače a servery. Poskytuje základní funkcionality jazyka Java od základních datových typů až po třídy umožňující přístup k databázím, práci se sítí, vytváření grafického uživatelského rozhraní (zejména za pomoci knihoven AWT a SWING) a práci s XML. Jedná se patrně o nejrozšířenější edici. [15] [24]

6.3. Java EE

Java *Enterprise Edition*, zkráceně Java EE, je platforma umožňující vytváření rozsáhlých podnikových síťových aplikací. Aplikace založené na této platformě se vyznačují především schopností zpracovávat současně více požadavků od více uživatelů, vysokým zabezpečením a integrací s databázovými systémy a jinými technologiemi. Představuje nadstavbu Java SE. [12] [24]

[18] API - *application programming interface* = souhrn softwarových komponent umožňující vytváření dalších komponent nebo aplikací.

7. Vybrané prvky jazyka Java

V následujících kapitolách budou popsány vybrané syntaktické a jiné prvky programovacího jazyka Java. Záměrně zde budou vynechány ty nejjednodušší části, mezi které můžeme zařadit datové typy, operátory, cykly, příkazy větvení a objasněny budou složitější a komplexnější elementy.

7.1. Třídy

Jak již bylo několikrát zmíněno, Java je objektový jazyk a právě třídy umožňují vytváření objektů. Každá třída vytváří nový datový typ, který lze po jeho deklaraci dále využívat. Třídy jsou ve své podstatě šablony objektů a instance tříd jsou samotnými objekty. V praxi se instance třídy a objekt velice často používají jako synonyma. Třídy sami o sobě nejsou uloženy v paměti, protože představují pouze určitou abstrakci. Do paměti se ukládají až jejich instance (objekty). [3] [6] [7]

Každá třída se skládá ze dvou částí: z atributů a metod. Atributy třídy představují data objektu a metody třídy obsahují kód, který pracuje s daty uloženými v atributech. Data jednotlivých objektů jsou tedy mezi sebou navzájem oddělená. [3] [6] [7]

Deklarace třídy se provádí pomocí klíčového slova *class*. Podle konvencí Javy se názvy tříd píší s velkým počátečním písmenem a název by měl vyjadřovat jednotné číslo. V případě víceslovného názvu se nepoužívají mezery, ale pro oddělení slov se používá, tzv. *Camel case* – každé nové slovo začíná velkým písmenem. Fyzicky jsou třídy ukládány na disk do souborů, jejichž název se přesně shoduje s názvem třídy a má příponu *.java*.

[3] [6] [7]

Obecný tvar třídy lze zapsat následujícím způsobem:

```
modifikátor přístupu class NázevTřídy {
        atributy
        konstruktor
        metody
}
```

Pro vytvoření instance třídy se používá klíčové slovo new. Způsob použití klíčové slova new a ukázka vytvoření instance dané třídy je uvedena níže:

```
NazevTřídy názevInstance = new NazevTřídy();
```

7.1.1. Abstraktní třída

Jednou ze zvláštních typů tříd je právě abstraktní třída. Deklaruje se klíčovým slovem *abstract* před klíčovým slovem *class*. Zvláštností této třídy je to, že může obsahovat abstraktní metody. Tyto metody jsou v abstraktní třídě deklarovány pouze jako signatura s návratovým datovým typem. Dále mohou obsahovat i přímou implementaci metod nebo libovolné definice atributů. U abstraktních tříd nelze vytvářet žádné jejich instance. Důvodem je to, že instance těchto tříd by prakticky neměli žádný smysl, protože velice často neobsahují implementaci jednotlivých metod. [5] [6] [7]

Všechny podtřídy abstraktní třídy musí implementovat všechny abstraktní metody uvedené v nadtřídě. Pokud by se tak nestalo je nutné i podtřídu označit jako abstraktní.

[5] [6] [7]

7.1.2. Vnitřní třída

Uvnitř tříd je možné deklarovat další třídy, které se označují jako vnitřní třídy. Vnitřní třídy mohou být umístěny i pouze lokálně v rámci nějakého bloku. Rozsah platnosti vnitřní třídy je omezen rozsahem vnější třídy. Vnitřní třída nemůže existovat nezávisle na vnější třídě. Vnitřní třídy mají přístup ke všem atributům a metodám vnější třídy, dokonce i k těm, které jsou deklarovány jako *private* a může se na ně přímo odkazovat. Opačný přístup ovšem možný není. Vnější třída nemá přístup k atributům vnitřní třídy. Instance vnitřních tříd může vytvářet pouze vnější třída. Pokus o vytvoření instance z jiné třídy, než která obsahuje vnitřní třídu, by vedl k chybě. [6] [7]

7.1.3. Dědičnost u tříd

V rámci dědičnosti rozděluje třídy na nadtřídy a podtřídy. Nadtřída je třída, od které dědí atributy a metody podtřída. Výjimkou jsou atributy s modifikátorem přístupu *private*, které se nedědí. Pro dědičnost se používá klíčové slovo *extends*, které se uvádí za názvem podtřídy a za tímto klíčovým slovem musí následovat název nadtřídy. Jazyk Java nepodporuje přímou dědičnost od více nadtříd, z čehož vyplývá, že každá podtřída musí mít pouze jednu přímou nadtřídu. Dědičnost u tříd ovšem můžeme vytvářet hierarchicky, což neodporuje předchozímu tvrzení. [6] [7]

7.2. Metody

Metody jsou společně s atributy základními součástmi tříd. Metody umožňují manipulaci s daty, poskytovat řízený přístup k datům a komunikaci s ostatními částmi programu. Dle konvencí se názvy metod píší s malými počátečními písmeny a měly by vystihovat to, co daná metoda skutečně provádí. Název nesmí obsahovat klíčová slova Javy a název *main(), který* je vyhrazen pro metodu zahajující provádění programu. Obecný tvar metody lze zapsat takto: [6] [7]

modifikátor_přístupu návratový_typ názevMetody(seznam parametrů){
tělo metody
}

7.2.1. Konstruktory

Konstruktor lze považovat za speciální typ metody, který má stejný název jako třída a slouží k inicializaci objektů při jejich vytvoření. Konstruktory nemají žádný návratový typ. Každá třída může mít libovolné množství přetížených konstruktorů nebo nemusí obsahovat konstruktor žádný. V tomto případě bude třídě automaticky přidělen defaultní konstruktor. Konstruktory, stejně jako metody, mohou být parametrizované.

[6] [7]

7.2.2. Přetěžování metod

Přetěžování metod představuje jeden ze způsobů, jakým Java implementuje polymorfismus. V rámci jedné třídy mohou existovat metody se stejným názvem, ale musí se lišit signaturou. Musí se tedy lišit počtem parametrů nebo datovými typy parametrů. Tyto metody se pak označují jako přetížené. Přetěžovat lze i konstruktory. [6] [7]

7.2.3. Přepisování metod

V rámci dědičnosti je možné, aby některá z podtříd měla stejnou signaturu a návratový typ metody jako metoda z nadtřídy. V tomto případě se nebude jednat o chybu, ale o přepsání (tzv. *override*) metody. Pokud se překrytá metoda zavolá z podtřídy, zavolá se metoda definovaná podtřídou. V případě, že se zavolá z nadtřídy, zavolá se původní nepřepsaná metoda. [6] [7]

7.3. Modifikátory přístupu

Modifikátory přístupu se používají pro řízení přístupu k jednotlivým členům, jako jsou například atributy, metody nebo i třídy. Existují čtyři modifikátory přístupu: žádný, *private*, *protected* a *public*. Jejich význam a možnosti přístupy nejlépe ilustruje následující tabulka: [6] [7]

	PRIVATE	**ŽÁDNÝ**	**PROTECTED**	**PUBLIC**
Viditelný v rámci stejné třídy	ANO	ANO	ANO	ANO
Viditelný podtřídou v rámci téhož balíčku	NE	ANO	ANO	ANO
Viditelný v rámci téhož balíčku třídou, která není podtřídou	NE	ANO	ANO	ANO
Viditelný podtřídou v rámci odlišného balíčku	NE	NE	ANO	ANO
Viditelný v rámci odlišného balíčku třídou, která není podtřídou	NE	NE	NE	NE

Tabulka 1: Modifikátory přístupu [6]

7.4. Balíčky

Balíčky poskytují možnost seskupení tříd, které spolu nějakým logickým způsobem souvisí, do jednoho většího celku. K jednotlivým třídám se potom přistupuje přes název balíčku za pomoci tečkové notace. Druhým významem balíčků je řízení přístupu k třídám, které obsahují (viz. Tabulka 1: Modifikátory přístupu). Seskupení tříd do pojmenovaných balíčků má ještě jeden význam – opětovná použitelnost názvů tříd. Názvy tříd totiž musí být v rámci jednoho balíčku unikátní. Pro deklaraci balíčku stačí na úplný začátek souboru doplnit klíčové slovo *package* následující názvem balíčku a středníkem. Již existující balíček je možné importovat do souboru pomocí klíčového slova *import* a názvu balíčku.

[6] [7]

7.5. Rozhraní

Rozhraní v Javě představuje mechanismus, jakým lze definovat, co mají třídy vykonávat, ale ne jakým způsobem. Z hlediska syntaxe jsou rozhraní podobná abstraktním třídám, ale s tím rozdílem, že rozhraní může obsahovat pouze hlavičky metod a statické konstanty. Všechny metody a statické konstanty deklarované v rozhraní jsou vždy veřejné, i bez ohledu na to, jestli je u nich použit modifikátor přístupu *public*. [5] [6] [7]

Rozhraní se definují prakticky stejně jako třídy, pouze se změní klíčové slova *class* na *interface*. Jedno rozhraní může implementovat jedno či více tříd. V tomto ohledu neexistuje žádné omezení. Rozhraní mohou také využívat dědičnosti. Stačí za název rozhraní přidat klíčové slova *extends* následované názvem rozhraní, od kterého se má dědit. Dědit může pouze rozhraní od jiného rozhraní, nikoli od jiné třídy. [6] [7]

7.5.1. Implementace rozhraní

Třída, u které potřebujeme implementovat nějaké rozhraní, musí za svým názvem obsahovat klíčové slovo *implements*, za kterým následuje název rozhraní, které třída bude implementovat. V těle třídy musí být následně doplněna všechna těla metod, která byla deklarována v rozhraní. Jedna třída může přímo implementovat několik rozhraní najednou, na rozdíl od dědičnosti, kde třída musí mít pouze jednu přímou nadtřídu. [6] [7]

7.6. Generické typy

Generické typy byly do Javy přidány ve verzi JDK 5 a jazyk změnily dvěma zásadními způsoby. První změnou bylo přidání nových syntaktických prvků, jako jsou například špičaté závorky, a druhá změna spočívala v úpravě mnoha tříd a metod v základním API. Generické typy umožňují vytvářet třídy, rozhraní a metody, které fungují typově bezpečně a zároveň jsou schopné pracovat s různými druhy dat. [6] [7]

Samotný pojem generické typy v podstatě znamená parametrizované typy. Generické typy umožňují vytvářet třídy, rozhraní a metody, u nichž se typ dat předává formou parametru. Výhodou generických typů je i automatický a implicitní převod datových typů. Parametry generických typů se uzavírají do špičatých závorek a mohou obsahovat konkrétní datový typ (zejména v případě kolekcí) nebo pouze typový parametr. Typových parametrů může býti více než jeden, stačí je oddělit čárkami. [6] [7]

Velmi důležitou vlastností generických typů je fakt, že pracují pouze s objekty. V případě použití některého z primitivních datových typů (např. int, double, float,char, short atd.) při vytváření instance generického typu dojde k chybě. Jako náhradu primitivních datových typů lze využít jejich obalové typy (Integer, Double, Char atd.).

[6] [7]

7.6.1. Omezení typových parametrů

V některých případech je nutné definovat určitá omezení typových parametrů. Zejména u různých matematických operací se vždy pracuje jen s čísly bez ohledu na to, jakého konkrétního datového typu jsou. Právě pro tyty situace Java nabízí omezení typových parametrů generických typů. Samotné omezení může vypadat následovně: [6] [7]

<T extends Number>

T je typový parametr, který zastupuje konkrétní datový typ deklarovaný při inicializaci generického typu. Klíčové slovo *exteds* říká, že datový typ T musí být podtřída *Number* nebo některá z jeho podtříd. [6] [7]

7.6.2. Zástupné argumenty

V případě generických typů je možné používat i zástupné argumenty, které se někdy také nazývají žolíky (*wildcard argument*). Účelem zástupným argumentů je v podstatě deklarace neznámého parametru, což je v některých případech užitečné. Jako symbol pro zástupný otazník se používá znak *?* uzavřený ve špičatých závorkách. [6] [7]

7.6.3. Omezení zástupných argumentů

Stejně jako „běžné“ parametry generických datových typů lze i zástupné argumenty omezovat. Oproti „klasickým“ parametrům mají ještě navíc tu výhodu, že je jim lze přidat nejen horní omezení, ale i dolní omezení. Pro horní omezení se používá klíčové slovo *extends* následované nadtřídou a pro dolní omezení se používá klíčové slovo *super* následované podtřídou. Podtřída však není do omezení zahrnuta, nadtřída ano. Ukázky obou omezení: [6] [7]

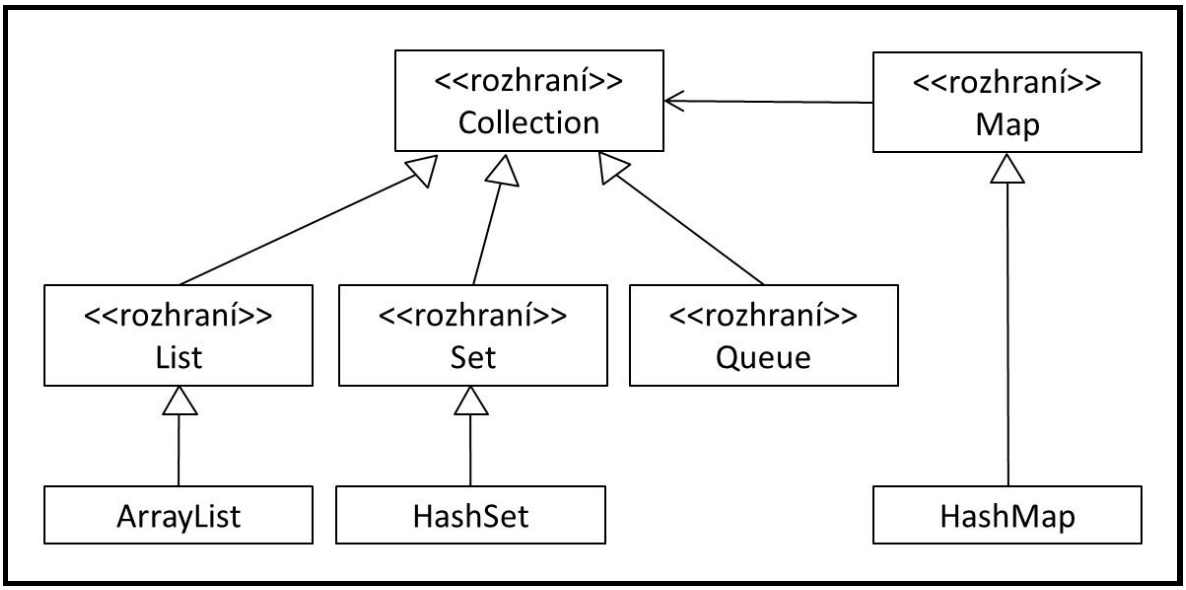
<? extends nadtřída>

<? super podtřída>

7.7. Kolekce

Kolekce jsou dynamické kontejnery, které mohou měnit počet svých prvků. Všechny kolekce jsou součástí tzv. *Collections Framwork*, který standardizuje způsob zpracování kolekcí objektů. Tento *framework* byl do Javy přidán až ve verzi J2SE 1.2. Před touto verzí Javy se pro seskupení objektů používaly třídy *Dictionary*, *Vector*, *Stack* a *Properties*. Dnes jsou tyto třídy označeny za zastaralé a nepoužívají se. [2] [5] [7]

Všechny kolekce jsou uloženy v balíčku *java.util* a implementují rozhraní *Collection*. Výjimku tvoří mapy, které implementují rozhraní *Map*, ale dokáží poskytnout i rozhraní *Collection*. Základní typy rozhraní a jejich implementujících tříd ilustruje následující obrázek: [2] [5] [7]



Obrázek 1: Základní typy kolekcí a jejich rozhraní [2]

Mezi hlavní výhody kolekcí můžeme zařadit hned několik jejich vlastností. V první řadě snižují množství programového kódu, protože vše potřebné je již naprogramované. Dále zrychlují program, protože jsou naprogramovány tak, aby výsledný kód byl co nejrychlejší. Kolekce zvyšují přehlednost a čitelnost programu, protože samotní programátoři nemusejí vymýšlet a vytvářet vlastní struktury. [2]

Samozřejmě u kolekcí lze nález i několik nevýhod. První z nich je fakt, že neumí ukládat primitivní datové typy a musí se použít jejich obalové typy. Kolekce jsou ve srovnání s obyčejnými poli ve většině případů pomalejší. [2]

7.7.1. Rozhraní Collection

Základem celého *Collections Framework* je právě generické rozhraní *Collection*, které musí implementovat všechny třídy definující konkrétní kolekci. Všechny kolekce je možné procházet rozšířeným cyklem *for*, protože rozhraní *Collection* přímo rozšiřuje rozhraní *Iterable*. [2] [7]

Základní metody definované tímto rozhraním lze posléze použít u všech konkrétních typů kolekcí, jako jsou například množiny a seznamy. Nejdůležitější metody jsou uvedené v následující tabulce: [2] [7]

boolean add (E obj)	Vloží objekt (prvek) do kolekce a vrací *true* nebo *false*. *True* v případě kladně vyhodnocené podmínky.
boolean addAll(Collection<? Extends E> c)	Do volající kolekce přidá všechny prvky z kolekce c.
void clear()	Vymaže všechny prvky z kolekce.
boolean contains(Object obj)	Testuje, zda-li je prvek v kolekci
boolean containsAll(Collection<?> c)	Testuje, jestli kolekce obsahuje všechny prvky obsažené v kolekci c.
boolean remove(Object obj)	Odstraní prvek z kolekce a vrátí výsledek akce.
boolean removeAll(Collection<?> c)	Z kolekce odstraní všechny prvky uložené v kolekci c.
boolean retainAll(Collection<?> c)	V kolekci ponechá všechny prvky z kolekce c. Ostatní prvky smaže.
int size()	Vrací počet prvků uložených v kolekci
boolean equals(Object obj)	Porovnává shodu kolekce s objektem obj.
int hashCode()	Vrací *hash* kód kolekce.
Object toArray()	Vrací pole, které obsahuje všechny prvky z kolekce.

Tabulka 2: Nejdůležitější metody z rozhraní Collection [2] [7]

7.7.2. Rozhraní List

List představuje rozhraní pro vytváření kolekcí nazývaných jako seznamy, které slouží k uchovávání posloupnosti prvků. Prvky mohou být vkládány přímo pomocí indexů, přičemž samotné indexování začíná od nuly. Seznamy udržují prvky seřazené a mohou obsahovat duplicitní hodnoty, protože se liší pozicí, na které jsou v kolekci uloženy.

[2] [5] [7]

7.7.3. Třída ArrayList

Třída *ArrayList* je nejpoužívanější třídou implementující rozhraní *List*. Instance této třídy jsou často využívány k tvorbě dalších druhů kolekcí. Nevýhodou instancí těchto tříd je nízká efektivita při častém vyzvedání a vkládání prvků doprostřed seznamu nebo i na začátek velkých seznamů. Pro tyto účely je efektivnější instance definovat jako seznam typu *LinkedList*. [2] [5] [7]

ArrayList má kromě své velikosti (počtu prvků uložených v kolekci) také kapacitu, kterou si lze představit jako předdefinovanou velikost kolekce. Java se automaticky stará o alokaci kapacity a v případě vyčerpání se kapacita zvětší o jeden a půl násobek původní velikosti. Pokud dopředu známe nebo dokážeme odhadnout, kolik prvků se do kolekce bude ukládat, můžeme jako parametr konstruktoru zadat kapacitu ručně. Tím zrychlíme chod programu. Vyhneme se tak vytvoření nové kolekce s větší kapacitou a následným překopírováním všech prvků z původní kolekce do nové, které Java automaticky provede po překročení kapacity. [2] [5] [7]

7.7.4. Rozhraní Set

Rozhraní *Set* tvoří naprostý základ pro kolekce typu množina. Tento druh kolekcí se vyznačuje tím, že stejné prvky v množině mohou být obsaženy pouze jednou. Vychází to i z matematické definice množiny. Kolekce typu množina tedy automaticky zabraňuje vzniku duplicitních hodnot. V porovnání s neseřazenými seznamy mají množiny obecně větší rychlost vyhledávání. Naopak nevýhodou množin je fakt, že neuchovávají pořadí prvků, v jakém byly do množiny ukládány. [2] [5] [7]

7.7.5. Třída HashSet

Třída *HashSet* je jednou z konkrétních implementací rozhraní *Set* a v praxi patří k nejčastěji využívaným kolekcím pro uložení množin. Ukládání dat do kolekce *HashSet* probíhá prostřednictvím *hash* tabulek, které obsahují *hash* kódy jednotlivých prvků kolekce. *Hash* kódem se rozumí celočíselná hodnota, která je automaticky generovaná a souvisí s určitým prvkem kolekce. Tyto kódy jsou pak uloženy do *hash* tabulky. *Hash* kód by ideálně měl být unikátní, ale může se stát, že dva objekty mohou vracet stejný *hash* kód. [2] [5] [7]

7.7.6. Rozhraní Queue

Rozhraní *Queue* představuje základ pro kolekce typu fronta. Fronta lze označit jako seznam, u kterého platí pravidlo *FIFO* (*First In, First Out*). První prvek vložený do kolekce typu fronta bude vyzvednut až jako poslední. Existují i jiné typy front, ve kterých se prvky řadí podle jiných kritérií, jako jsou například prioritní fronty. Jak už napovídá sám název, prvky jsou řazeny podle priority. Fronty mají oproti jiným kolekcím jednu ojedinělou vlastnost. Prvky je možné odebírat pouze ze začátku fronty. Samotné rozhraní *Queue* deklaruje několik metod pro vkládání prvků do fronty a odebírání prvků z fronty.

[2] [5] [7]

7.7.7. Rozhraní Deque

Rozhraní *Deque* implementuje rozhraní *Queue* a deklaruje metody pro oboustranné fronty. Oboustranné fronty se mohou chovat jako „klasické“ fronty s organizací *First In, First out* nebo se mohou chovat jako zásobníky s organizací *LIFO* (*Last In, First Out*). Poslední vložený prvek do fronty bude vyzvednut (načten) jako první. Kapacita oboustranné fronty může být omezena a do fronty potom lze přidat pouze omezený počet prvků. [5] [7]

7.7.8. Rozhraní Map

Mapy jsou zvláštním typem kolekcí, které jsou někdy označovány i jako slovníky. Prvek mapy tvoří vždy dvojice objektů nazývaných klíč a hodnota. Klíč musí být vždy unikání a slouží k vyhledávání hodnot. Hodnoty mohou být duplicitní. Všechny mapy musejí implementovat rozhraní Map, které deklaruje metody pro práci s mapami. Mezi konkrétní třídy implementující toto rozhraní patří například *HashMap*. [2] [5] [7]

8. Zpracování výjimek

Zpracování výjimek patří k nejdůležitějším částem každého programu. Bez ošetření výjimečných, respektive chybových stavů, dojde k pádu aplikace. Výjimku tedy lze chápat jako běhovou chybu. Výjimka je v Javě reprezentována objektem popisujícím chybový stav, který vznikl v nějaké části kódu. Metoda, ve které byla výjimka vyvolána, obdrží tento objekt, který buď sama zpracuje, nebo jej předá dál ke zpracování. [5] [7]

8.1. Typy výjimek

Na vrcholu hierarchie výjimek se nachází třída *Throwable*, od které dědí všechny ostatní třídy. Jejími přímými podtřídami jsou třídy *Exceotion* a *Error*, které výjimky rozdělují do dvou odlišných skupin. Třída *Exception* a zejména její podtřída *RuntimeException* představují chybové stavy, které by měly být zachytávány uživatelskými aplikacemi. Třída *Error* definuje skupinu výjimek, u kterých se nepředpokládá jejich ošetřování v uživatelských aplikacích. Výjimky tohoto typu souvisejí přímo s běhovým prostředím Javy a v běžných aplikacích se jimi programátoři vůbec nezabývají. [5] [7]

8.2. Příkazy pro ošetření výjimek

Ke zpracování výjimečných stavů se v Javě používá celkem pět klíčových slov, konkrétně *try, catch, finally, throw* a *throws*. Jejich význam je popsán níže. [5] [7]

8.2.1. Blok try

Veškerý kód, ve kterém chceme zachytávat chybové stavy, musí být uzavřen v bloku *try*. Tyto bloky je možné do sebe zanořovat. V případě, že některý z vnitřních bloků *try* neobsahuje žádný příkaz *catch*, běhové prostředí se pokusí najít shodu ve vnějších blocích. Pokud ani tam shodu nenajde, pak zpracování zajistí samotné běhové prostředí. [5] [7]

8.2.2. Blok catch

Blok *catch* následuje bezprostředně za blokem *try* a specifikuje typ výjimek, které chceme zpracovávat. Některé části kódu mohou vyvolávat více typů výjimek, a proto je možné použít více příkazů *catch*, které zpracovávají jednotlivé typy chyb. [5] [7]

8.2.3. Blok finally

Jedním z problémů při vzniku běhových chyb je fakt, že se mění provádění kódu metody. U některých metod může toto chování vést k dalším chybám. K zabránění těchto nechtěných stavů slouží právě blok *finally*. Kód obsažený v tomto bloku se provede bezprostředně po vykonání kódu z bloku *catch*, který zachytil výjimku. Kód z bloku *finally* se ale provede i v případě, kdy nedošlo k zachycení žádné výjimky. Provádí se tedy vždy bez ohledu na vyvolání či nevyvolání výjimky. [5] [7]

8.2.4. Příkaz throw

V některých případech je užitečné vyvolat výjimky ručně. K explicitnímu vyvolání výjimek se používá právě příkaz *throw* následovaný typem výjimky, kterou chceme vyvolat. Po zavolání tohoto příkazu okamžitě končí vykonávání programu, což znamená, že žádný následující příkaz nebude proveden. Dále se zkontrolují obklopující bloky *try*, zda-li neobsahují blok *catch* odpovídající typu vyvolané výjimky. V případě nenalezené shody se o obsluhy výjimky postará běhové prostředí Javy. [5] [7]

8.2.5. Klauzule throws

Klauzule *throws* se přidává do deklarace metody, která je schopna vyvolat výjimky, ale sama je nezpracovává. Typy výjimek, které metoda může vyvolat, jsou vyjmenovány právě v klauzuli *throws*. Výjimky typu Error, RuntimeException a jejich podtřídy se ve výčtu výjimek klauzule *throws* neuvádějí. [7]

8.3. Vlastní podtřídy výjimek

Standardní výjimky jazyka Java umožňující obsluhu většiny běžných chyb. Pro některé specifické aplikace či situace může být žádoucí vytvoření vlastních typů výjimek. Vlastní typ výjimky lze vytvořit pouze rozšířením třídy *Exception*, tedy vytvořením její podtřídy (za pomoci klíčového slova *extends*). Podtřídy nemusejí implementovat žádný konkrétní kód, protože jejich pouhá existence v typovém systému postačuje k používání vlastních typů výjimek. [7]

Při tvorbě vlastních typů výjimek bychom měli dbát na to, aby název výjimky i její popis skutečně vystihoval chybový stav a nepůsobil spíše matoucím dojmem. [5]

9. Grafické uživatelské rozhraní

Všechny současné aplikace uživatelům nabízejí určité grafické rozhraní pro komfortnější a jednodušší ovládání. Jazyk Java programátorů nabízí dvě knihovny pro tvorbu grafického uživatelského rozhraní, konkrétně knihovnu AWT a Swing.

9.1. Abstract Window Toolkit (AWT)

Knihovna AWT obsahuje velké množství tříd a metod sloužících k vytváření a správě oken a poskytuje základy pro knihovnu Swing. Je uložena v balíčku *java.awt,* který patří mezi největší balíčky Javy. Omezením této knihovny je fakt, že své vizuální komponenty překládá na ekvivalenty specifické pro konkrétní platformu. Z toho vyplývá, že vzhled neurčuje sama Java, ale platforma. Komponenty knihovny AWT jsou označovány jako těžké (*heavyweight*), protože využívají prostředky nativního kódu. [7]

9.1.1. Třída Component

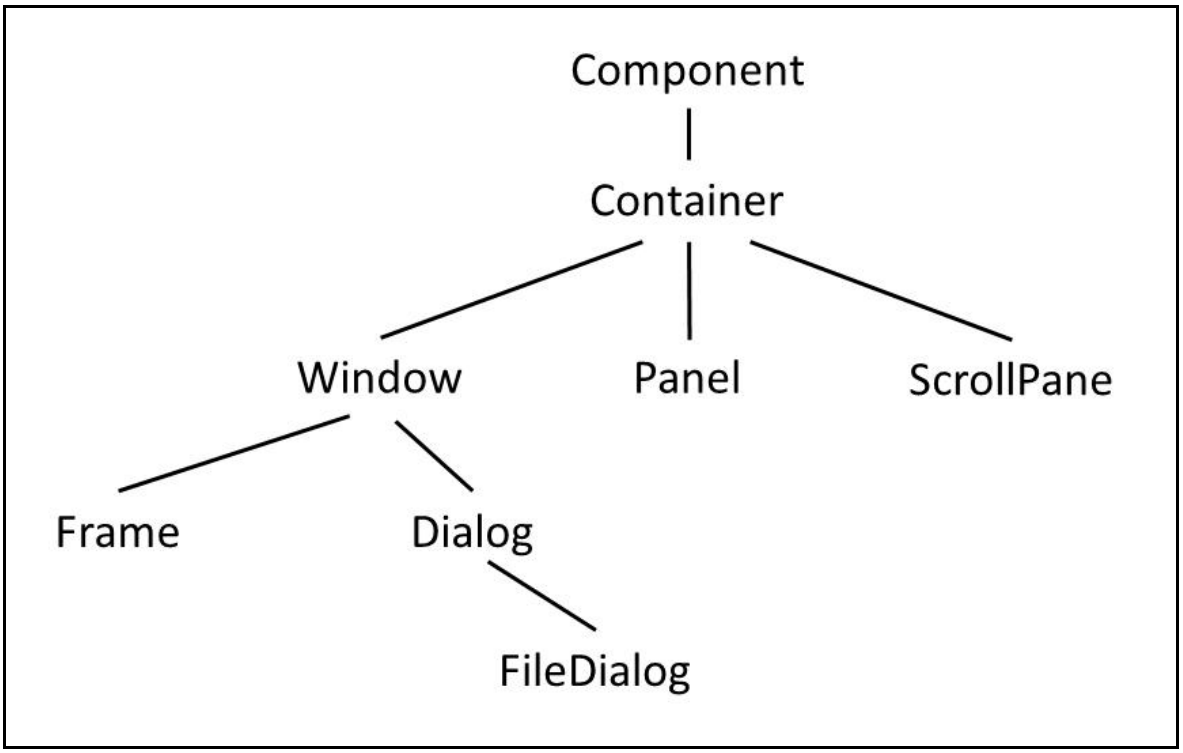
Knihovna AWT definuje hierarchii tříd, na jejímž vrcholu se nachází abstraktní třída *Component*, od které jsou zděděny veškeré další komponenty. Další úrovně hierarchie přidávají funkcionalitu a zvyšují specifičnost komponenty. Zapouzdřuje všechny atributy vizuální komponenty a definuje více než stovku veřejných metod. [1] [7]

9.1.2. Kontejnery

Všechny komponenty musí být umístěny v nějakém kontejneru, aby je bylo možné zobrazit. Kontejner je tedy komponenta, která může obsahovat další komponenty nebo jiné kontejnery. Nejvyšší třídou pro hierarchii kontejnerů je třída *Container*, která dědí od třídy *Component* a obsahuje metody pro vkládání komponent. [1] [7]

Přehled všech kontejnerových komponent, které dědí z třídy *Container*: [7] [7]

- **Window** – okno nejvyšší úrovně, které není obsaženo v jiném objektu.
- **Frame** – podtřída Window. Okno se záhlavím a lištou s nabídkou.
- **Panel** – okno bez ohraničení, záhlaví a lišty.
- **ScrollPane** – umožňuje vytvořit posouvací plochu.
- **Dialog** – dialogové okno s titulkem a okraji.
- **FileDialog** – dialogové okno pro výběr souborů, neobsahuje další prvky.



Obrázek 2: Struktura kontejnerů v AWT [7] [7]

Vzhledem k tomu, že se v praxi používá a je doporučováno, aby se používala knihovna Swing, nebudou zde popsány další komponenty z knihovny AWT, jako například popisky, tlačítka, zaškrtávací pole a další. Navíc knihovna AWT je natolik rozsáhlá, že popis všech jejich komponent je nad rámec této bakalářské práce.

9.2. Swing

Knihovna *Swing* byla vytvořena jako reakce na nedostatky knihovny AWT, na jejíž základech je založena. *Swing*, až na některé výjimky, využívá lehké (*lightweight*) komponenty a podporuje připojitelný vzhled o ovládání (*Pluggable Look And Feel*). [6] [7]

Lehké komponenty jsou celé napsány v Javě a nemapují se na prostředky nativního kódu dané platformy, čímž je zajištěn stejný vzhled na všech platformách. Díky této skutečnosti jsou lehké komponenty efektivnější a flexibilnější než těžké. [6] [7]

Swing odděluje vzhled a chování všech komponent, díky čemuž je možné připojit nový vzhled komponenty, aniž by to mělo dopad na její chování. Tato vlastnost funguje i opačně. Lze změnit chování komponenty, aniž by se změnil její vzhled. Díky této vlastnosti lze definovat vzhled stejný pro všechny platformy, ale i vzhled specifický pro konkrétní platformu. [6] [7]

9.2.1. Kontejnery

Kontejnery knihovny *Swing* lze rozdělit na dva základní typy. Prvním typem jsou kontejnery nejvyšší úrovně (těžké kontejnery), které jsou rozšířením tříd *Component* a *Container* z knihovny AWT. Mezi tyto kontejnery patří *JFrame, JApplet, JWindow* a *JDialog*. Kontejnery nejvyšší úrovně, jak již vyplývá z názvu, se musí nacházet na vrcholu celé hierarchie uspořádání komponent a nejsou součástí žádného jiného kontejneru. [6] [7]

Druhým typem jsou lehké kontejnery, které rozšiřují třídu *JComponent*. Nejčastěji používaným zástupcem tohoto typu je třída *JPanel*, která slouží k uspořádání vzájemně souvisejících komponent a tvorbě layoutů. [6] [7]

9.2.2. Komponenty

Všechny komponenty knihovny *Swing* jsou odvozeny od třídy *JComponent*, která podporuje připojitelný vzhled a ovládání. Samotná třída *JComponent* rozšiřuje třídy *Component* a *Container* z knihovny AWT. Všechny komponenty knihovny Swing jsou uloženy v balíčku *javax.swing*. Následující tabulka obsahuje přehled všech tříd reprezentujících komponenty knihovny *Swing*. [6] [7]

JApplet	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayer
JLayeredPane	JList	JMenu	JMenuBar
JMenuItem	JOptionPane	JPanel	JPasswordField
JPopupMenu	JProgressBar	JRadioButton	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane	JSeparator
JSlider	JSpinner	JSplitPane	JTabbedPane
JTable	JTextArea	JTextField	JTextPane
JToggleButton	JToolBar	JToolTip	JTree
Jviewport	JWindow		

Tabulka 3: Komponenty knihovny Swing [6] [7]

10. Desktopová aplikace Tester JSDH

Součástí této bakalářské práce je i praktická část, která je reprezentována desktopovou aplikaci nazvanou Tester JSDH. Primárním účelem aplikace je testování odborných znalostí členů jednotek dobrovolných hasičů nejen ve školicích střediscích, ale i přímo v jednotlivých sborech.

Mezi dílčí účely aplikace lze zařadit zrychlení a zautomatizování oprav testů, protože většina školicích středisek dodnes k ověření znalostí získaných v kurzech používá papírovou formu testů, jejichž oprava zabere instruktorům poměrně dost času. V neposlední řadě aplikace názorně ukazuje praktické použití vybraných prvků Javy, které byly popsány v teoretické části této práce.

Dále demonstruje způsob možného řešení pro aplikaci, která má sloužit pro ověření znalostí, tedy pro aplikaci generující náhodné otázky na základně předem zvoleného schématu. Tuto implementaci lze využít nejen v oblasti související s testováním členů jednotek dobrovolných hasičů, ale i ve všech ostatních případech, kdy je potřeba ověření znalostí z určité oblasti.

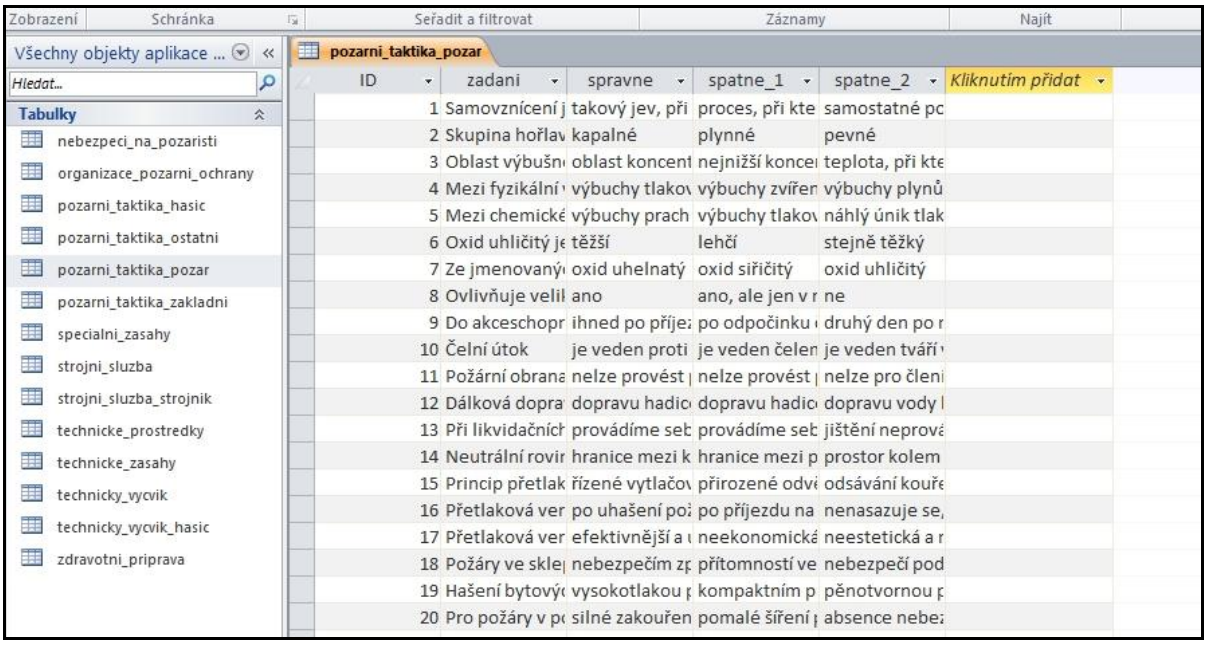
Aplikaci je možné spustit přímo přes vývojové prostředí *NetBeans IDE 8.1* nebo pomocí vygenerovaného JAR souboru.

10.1. Databáze otázek

Všechny testové otázky jsou uloženy v databázi vytvořené programem *Microsoft* Access v souboru *otazky_hasici.accdb*. Celá databáze je rozdělena do 14 tabulek, z nichž každá obsahuje otázky z jedné zkušební oblasti, která se váže k odpovídajícímu typu testu. Struktura všech tabulek je jednotná a je tvořena následujícími sloupci:

- **ID** – představuje unikátní číslo otázky v rámci tabulky
- **zadani** – celé znění testové otázky
- **spravne** – sloupec obsahující správnou odpověď na otázku
- **spatne_1** – sloupec obsahující první špatnou odpověď na otázku
- **spatne_2** – sloupec obsahující druhou špatnou odpověď na otázku

Testové otázky jsou reálné a odpovídají otázkám používaných ve školicích střediscích. Otázky pro kategorie *Strojník JPO* a *Velitel JPO* byly převzaty z webových stránek Hasičského záchranného sboru České republiky [9] a otázky pro kategorie Hasič I. – III. stupně byly převzaty z webových stránek Okresního sdružení hasičů Kutná Hora [18].



Obrázek 3: Struktura otázek v databázi

10.2. Popis implementace

Pro vývoj aplikace bylo zvoleno vývojové prostředí *NetBeans IDE 8.1*, které je bezplatně ke stažení ze stránek výrobce a jak již bylo zmíněno výše, pro vytvoření databáze byl zvolen program *Microsoft Access*. Celá aplikace je logicky rozdělena do čtyř balíčků, které umožňují snazší orientaci v celém projektu. Struktura balíčků a tříd, které obsahují je následující:

- **grafika** – balíček obsahující všechny třídy vykreslující grafické prvky
  - třídy: HlavniOkno.java a OknoTestu.java
- **logika** – balíček pro třídy související se samotnou logikou aplikace
  - třídy: FormatTestu.java, Otazka.java, PripojeniKAccessDB.java a Start.java
- **obrazky** – balíček pro seskupení obrázků použitých v aplikaci
- **otazky** – baliček pro soubory s otázkami

10.2.1. Třída HlavniOkno

Tato třída představuje úvodní oko, které se zobrazí po spuštění aplikace. Základem celé třídy je kontejner *JFrame*, jehož instance je nazvaná *hlavniOkno*, do které jsou přidávány další kontejnery typu *JPanel* s jednotlivými komponentami. V konstruktoru této třídy se volá několik metod pro nastavení vzhledu:

- **hlavniOkno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)** pro nastavení ukončení celé aplikace po stisknutí červeného křížku v pravém horním rohu okna
- **hlavniOkno.setTitle("Tester JSDH")** – nastaví oknu titulek zadaný jako parametr
- **hlavniOkno.setResizable(false)** – zamezí manuálnímu roztahování okna
- **hlavniOkno.setVisible(true)** – „zviditelní“ celé okno
- **hlavniOkno.add(nastavObsahOkna())** – přidá do okna další kontejnery a komponenty nastavené v metodě *nastavObsahOkna()*
- **hlavniOkno.pack() –** velikost okna se přizpůsobí velikost komponent, které obsahuje
- **hlavniOkno.setLocation(x,y) –** nastaví pozice pro zobrazení okna, proměnné *x* a *y* představují souřadnice

Zbylé metody této třídy slouží pouze pro nastavení instancí kontejnerů *JPanel*, které obsahují komponenty typu *JLabel*, které v podstatě reprezentují vše viditelné v okně a obsahují i obrázky. Všechny kontejnery používají jako správce rozvržení *BoxLayout*, kromě panelu v metodě *nastavObsahOkna()*, která využívá *GridLayout* umožňující rozmístění komponent do pravidelné mřížky.

Všechny typu testů jsou graficky reprezentovány instancí třídy *JLabel*, která obsahuje obrázek i textový popis. Všem těmto instancím je přiřazen vlastní *MouseListener*, který je implementován jako privátní vnitřní třída rozšiřující třídu *MouseAdapter* a implementující rozhraní *MouseListener*. V těle těchto vnitřních tříd jsou implementovány metody *mouseClicked()*, která spustí okno testu s otázkami po kliknutí na komponentu, a *mouseEntered()*, která změní typ kursoru po najetí nad komponentu.

10.2.2. Třída OknoTestu

Druhá grafická třída, která slouží pro vykreslení okna s předem vygenerovanými otázkami. Stejně jako předchozí třída je založena na instanci kontejneru *JFrame*, jehož vlastnosti a obsah se opět nastaví v parametrizovaném konstruktoru, který jako parametr přijímá objekt typu *String*, který specifikuje typ testu.

Grafickou podobu pro zadání otázek zajišťují instance třídy *JLabel* a pro jednotlivé možnosti odpovědí jsou využity instance třídy *JRadioButton*, které jsou přidávány do *ButtonGroup,* aby byla zajištěna možnost výběru pouze jedné možnosti z nabízených třech. Všechny tyto grafické komponenty jsou dynamicky generovány a ukládány do pole, které následně slouží k uložení vybraných odpovědí a vygenerovaných otázek.

Nejdůležitější částí této třídy je metoda *opravTest()*, která ve svém těle projde pole odpovědí. Zjistí, která možnost je zaškrtnutá a porovná vybranou odpověď se správnou odpovědí. V případě schody inkrementuje proměnnou *pocetSpravnychOdpovedi* a celý text podbarví zeleně. Pokud je vybrána špatná odpověď, tak se její text podbarví červeně a správná odpověď zeleně, aby byla možná následná revize testu. Jestli uživatel nezvolí žádnou z možností, pak se text správné odpovědi také podbarví zeleně. Opět z důvodu revize. Právě z důvodu podbarvování odpovědí je tato metoda umístěna v grafické třídě a ne v některých z logických tříd.

Časové omezení testu a grafické znázornění hodin, které se odečítají po vteřinách je implementováno v metodě *panelSHodinami()*, která využívá třídu *Timer* pro nastavení časového intervalu, který se zobrazuje v komponentně *JLabel*. V této metodě se nastaví počet minut a vteřin, které reprezentují číselné proměnné *min* a *sec*. Po zavolání metody *scheduleAtFixedRate()*, jejímž parametrem je vytvoření vnitřní třídy *TimerTask*. Ta obsahuje metodu *run(),* v jejímž těle se proměnná *sec* dekrementuje v časovém intervalu tisíc milisekund. V případě, že je *sec* rovna nule a *min* je větší než nula, pak se proměnná *sec* nastaví na hodnotu 59 a *min* se o jednu sníží. Pokud jsou obě proměnné rovny nule, zavolá se metoda *opravTest()*, nastaví se výsledek testu a zobrazí se okno informující o vypršení časového intervalu.

Pro ukončení testu před vypršením časového intervalu slouží tlačítko ve spodní části okna. Po jeho stisknutí se čas zastaví a test se opraví.

10.2.3. Třída FormatTestu

Třída obsahující metody pro generování čísel otázek pro jednotlivé typy testů podle zvoleného schématu. Vzhledem k tomu, že všechny typy testů mají předem definovány okruhy otázek, které reprezentují jednotlivé tabulky v databázi, je nutné zajistit výběr otázek z jednotlivých tabulek. Z tohoto důvodu obsahuje třída *formatTestu* konstanty reprezentující názvy tabulek a počty otázek v tabulkách.

Tyto konstanty jsou využívány v ostatních metodách této třídy. Pro generování náhodných čísel otázek se využívá metoda *getNahodnaCislaOtazek(int celkemOtazek, int pozadovanyPocetOtazek),* v jejímž těle se vytvoří *ArrayList<Integer>,* do kterého se za pomoci *for* cykly uloží čísla od jedné do čísla uloženém v proměnné *celkemOtazek*. Dále použita statická metoda *Collections.shuffle(kolekce)*, která náhodně prohází prvky uvnitř kolekce předané jako parametr. Posledním krokem je uložení požadovaného počtu čísel do objektu typu *String*, který slouží jako návratová hodnota metody.

Název tabulky a otázky, které se mají nahrát z databáze, jsou ukládány do mapy typu *HashMap<String, String>*, kde klíč představuje mapu a hodnota čísla otázek.

10.2.4. Třída Otazka

Třída *Otazka* reprezentuje jednu otázku testu. V konstruktoru se nastaví všechny atributy, které odpovídají názvům sloupců z databáze, a dále obsahuje metody pro získání hodnot všech atributů.

10.2.5. Třída PripojeniKAccessDB

Účelem této třídy je propojení desktopové aplikace s databází. Základem je ovladač *Ucanaccess* [22], který tuto komunikaci umožňuje. Tato třída obsahuje jedinou metodu *getOtazkyZDatabaze*, která vrací kolekci otázek a jako parametr přijímá mapu obsahující dvojici *název_tabulky:čísla_otázek*, kterou vloží do SQL dotazu, který vrátí data.

10.2.6. Třída Start

Jedná se o spouštěcí třídu celé aplikace obsahující metodu *main*, ve které se vytvoří instance třídy *HlavniOkno*.

10.3. Zhodnocení výsledků praktické části

Hlavním cílem praktické části této bakalářské práce bylo vytvoření desktopové aplikace, která bude umožňovat ověření znalostí členů jednotek dobrovolných hasičů. Naprogramovaná aplikace splňuje všechny níže uvedené funkcionality:

- testovací otázky jsou uloženy v databázi mimo aplikaci
- možnost výběru typu testu podle odbornosti nebo funkce v jednotce
- časové omezení testů
- automatické vyhodnocování a oprava testů
- možnost revize testů po jejich ukončení

Při tvorbě aplikace se vyskytl problém s ovladačem ODBC, který Java 8 přestala podporovat a komunikace s databází pomocí mostu JDBC-ODBC nebyla možná. K vyřešení tohoto problému byl použit ovladač *Ucanaccess*, který je v doposud poslední verzi Javy podporován.

Aplikace je plně funkční a byla otestována několika členy jednoho nejmenovaného sboru dobrovolných hasičů. Jejich hodnocení proběhlo velice kladně. Praktické nasazení aplikace do školicích středisek nebo přímo do sborů je zcela jistě možné, nicméně bylo by dobré rozšířit některé okruhy o další testové otázky.

11. Závěr

Programovací jazyk Java nabízí široké spektrum využití, které začíná u malých jednoduchých konzolových aplikací, pokračuje přes středně velké mobilní a desktopové aplikace a končí u rozsáhlých podnikových aplikací. S příchodem operačního systému Android, jehož základem je právě jazyk Java, se tento jazyk rozšířil na další zařízení. Mezi ty nejrozšířenější lze zařadit chytré mobilní telefony, Smart TV nebo Smart Watch.

Popularita Javy se několik let drží na předních příčkách, což dokazuje i index popularity programovacích jazyků (viz. Přílohy, Obrázek 7 a 8), který pravidelně vypočítává společnost TIOBE [21], a lze předpokládat, že tomu tak bude i nadále.

Při návrhu a realizaci aplikací nelze jednoznačně určit nejlepší možné řešení, protože způsobů samotné implementace je vždy hned několik a záleží jen na vývojářích, pro kterou možnost se rozhodnou. Doporučené způsoby implementace jsou dokumentovány vývojem konkrétní aplikace. Podle mého osobního názoru je jazyk Java robustním nástrojem pro vývoj malých, středních i velkých podnikových aplikací.

12. Seznam použitých zdrojů

12.1. Tištěné

1. HEROUT, Pavel. 2007. *Java: grafické uživatelské prostředí a čeština*. 2. vyd. České Budějovice: Kopp, 316 s. ISBN 978-80-7232-328-9

2. HEROUT, Pavel. 2008. *Java - bohatství knihoven*. 3. vyd. České Budějovice: Kopp, 251 s. ISBN 978-80-7232-368-5.

3. KISZKA, Bogdan. 2009. *1001 tipů a triků pro jazyk Java.* Vyd. 1. Brno: Computer Press, 542 s. ISBN 978-80-251-2467-3.

4. KOEGH, James. *Java bez předchozích znalostí: průvodce pro samouky*. Vyd. 1. Brno: Computer Press, 2005, 274 s. ISBN 80-251-0839-2.

5. PECINOVSKÝ, Rudolf. 2014. *Java 8: úvod do objektové architektury pro mírně pokročilé*. 1. vyd. Praha: Grada, 655 s. Knihovna programátora (Grada). ISBN 978-80-247-4638-8.

6. SCHILDT, Herbert. 2012. *Java 7: výukový kurz*. 1. vyd. Brno: Computer Press, 664 s. ISBN 978-80-251-3748-2.

7. SCHILDT, Herbert. 2014. *Mistrovství - Java: Kompletní průvodce vývojáře*. 1. vyd. Brno: Computer Press. ISBN 978-80-251-4145-8.

12.2. Online

8. GOSLING, James a Frank YELLIN. Java API Documentation [online]. [cit. 2015-09-14]. Dostupné z: http://web.archive.org/web/19990202194011/http://java.sun.com/products/jdk/1.0.2/api/

9. *Hasičský záchranný sbor České republiky* [online]. [cit. 2016-02-01]. Dostupné z: http://www.hzscr.cz/clanek/testove-otazky-k-odborne-zpusobilosti-clenu-sdh.aspx

10. *JAR File Specification* [online]. [cit. 2015-09-18]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html#JAR_Index

11. *Java Date-Time Packages* [online]. [cit. 2015-09-24]. Dostupné z: http://docs.oracle.com/javase/8/docs/technotes/guides/datetime/index.html

12. *Java EE at a Glance* [online]. [cit. 2015-10-05]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview/index.html

13. *JAVA PLATFORM, MICRO EDITION (JAVA ME)* [online]. [cit. 2015-10-05]. Dostupné z: http://www.oracle.com/technetwork/java/embedded/javame/index.html

14. *Java SE 8: Lambda Quick Start* [online]. [cit. 2015-09-24]. Dostupné z: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html

15. *Java SE at a Glance* [online]. [cit. 2015-10-05]. Dostupné z: http://www.oracle.com/technetwork/java/javase/overview/index.html

16. *Java Timeline* [online]. [cit. 2015-09-14]. Dostupné z: http://oracle.com.edgesuite.net/timeline/java/

17. KULANDAI, Joseph. *Java Versions, Features and History* [online]. [cit. 2015-09-14]. Dostupné z: http://javapapers.com/core-java/java-features-and-history/

18. *Okresního sdružení hasičů Kutná Hora* [online]. [cit. 2016-02-01]. Dostupné z: http://www.oshkh.cz/2011/07/vzorove-testove-otazky-pro-ziskani-odznaku-odbornosti-hasic- i-iii-stupne/

19. RAJPUT, Dinesh. *Java Versions, Features and History* [online]. [cit. 2015-09-14]. Dostupné z: http://www.dineshonjava.com/2013/01/java-versions-features-and-history.html#.VfaxNhHtlBe

20. STEJSKAL, Michal. *Novinky v Javě 8* [online]. [cit. 2015-09-24]. Dostupné z: http://blog.bcvsolutions.eu/novinky-v-jave-8/

21. *TIOBE: The software quality company* [online]. [cit. 2016-02-22]. Dostupné z: http://www.tiobe.com/tiobe_index?page=index

22. *UCanAccess* [online]. [cit. 2016-02-10]. Dostupné z: http://ucanaccess.sourceforge.net/site.html

23. *What's New in JDK 8* [online]. [cit. 2015-09-24]. Dostupné z: http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

24. *Your First Cup* [online]. [cit. 2015-10-05]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

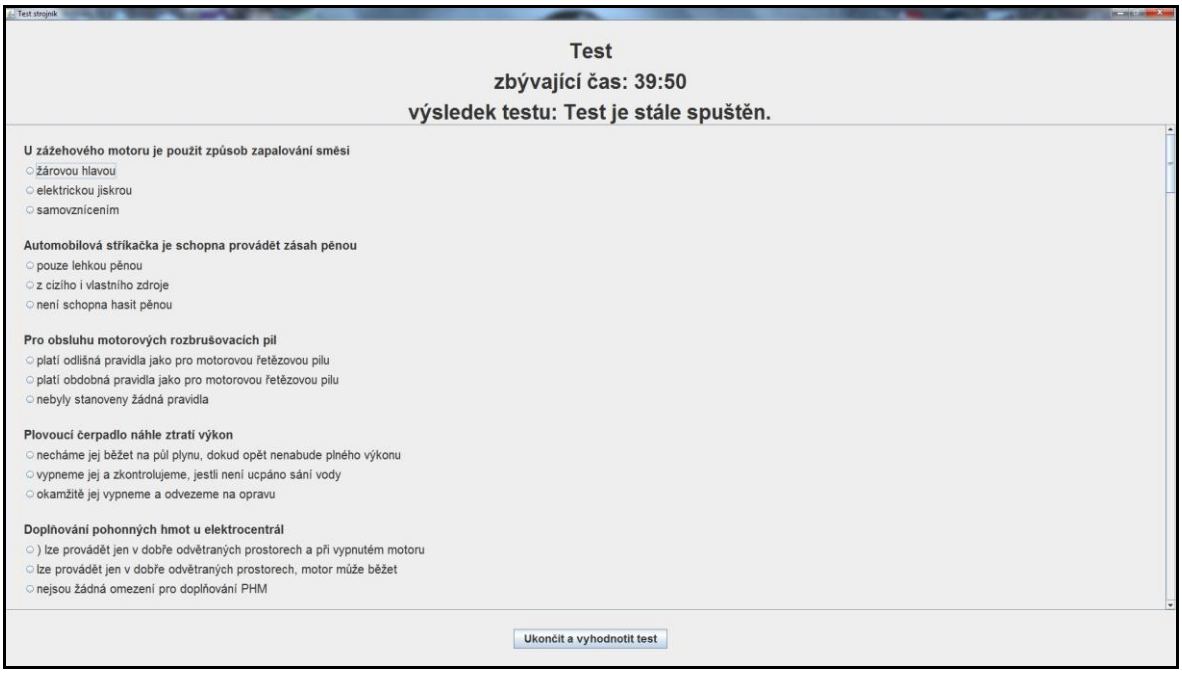
13. Přílohy

13.1. Seznam obrázků

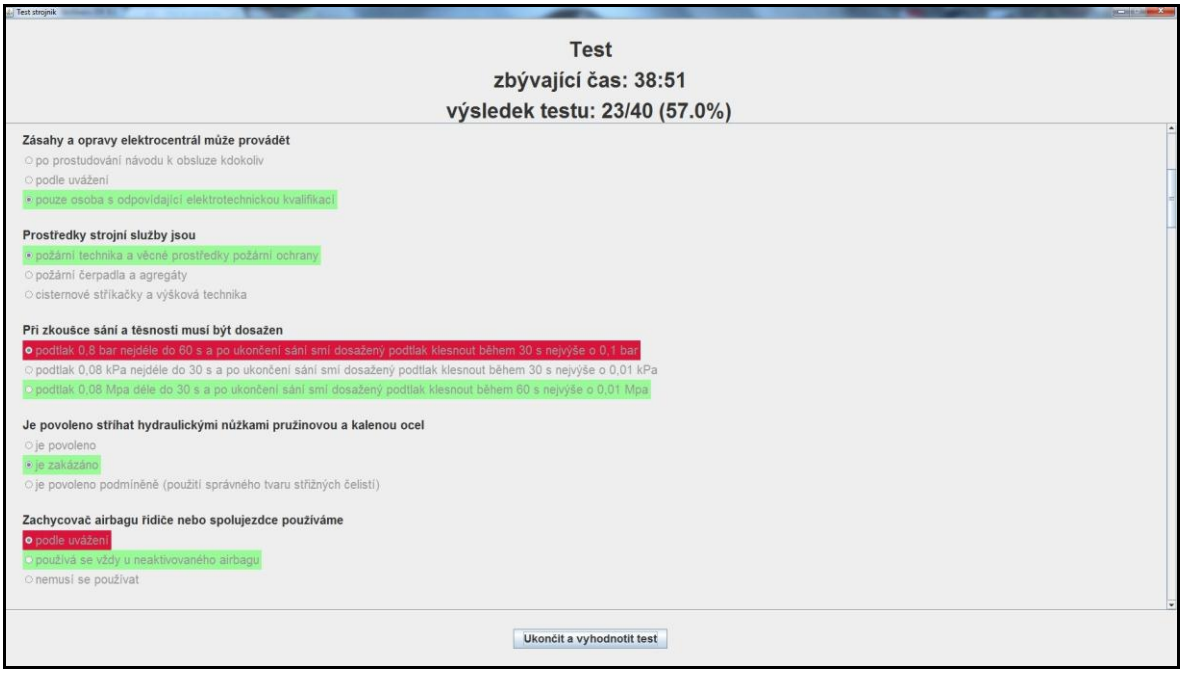
13.2. Seznam tabulek

13.3. Doplňující obrázky

Obrázek 4: Úvodní obrazovka aplikace



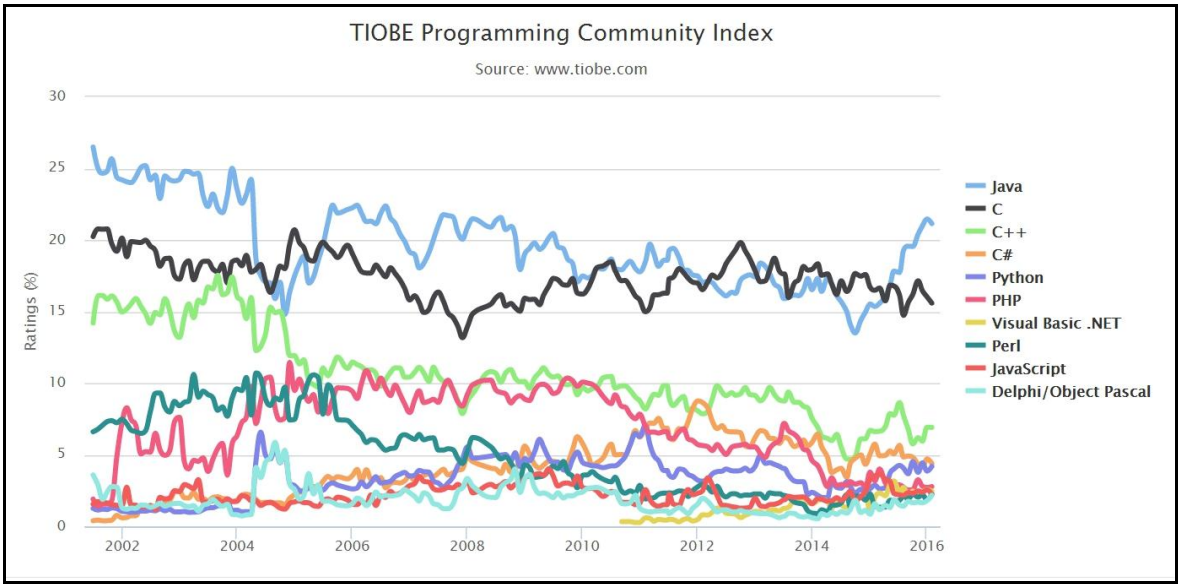
Obrázek 5: Ukázka vygenerovaného testu



Obrázek 6: Ukázka vyhodnoceného trestu

Feb 2016	Feb 2015	Change	Programming Language	Ratings	Change
1	2	↑	Java	21.145%	+5.80%
2	1	↓	C	15.594%	-0.89%
3	3		C++	6.907%	+0.29%
4	5	↑	C#	4.400%	-1.34%
5	8	↑	Python	4.180%	+1.30%
6	7	↑	PHP	2.770%	-0.40%
7	9	↑	Visual Basic .NET	2.454%	+0.43%
8	12	⇈	Perl	2.251%	+0.86%
9	6	↓	JavaScript	2.201%	-1.31%
10	11	↑	Delphi/Object Pascal	2.163%	+0.59%
11	20	⇈	Ruby	2.053%	+1.18%
12	10	↓	Visual Basic	1.855%	+0.14%
13	26	⇈	Assembly language	1.828%	+1.08%
14	4	⇊	Objective-C	1.403%	-4.62%
15	30	⇈	D	1.391%	+0.77%
16	27	⇈	Swift	1.375%	+0.65%
17	18	↑	R	1.192%	+0.23%
18	17	↓	MATLAB	1.091%	+0.06%
19	13	⇊	PL/SQL	1.062%	-0.20%
20	33	⇈	Groovy	1.012%	+0.51%

Obrázek 7: Tabulka indexů popularity programovacích jazyků [21]



Obrázek 8: Graf indexů popularity programovacích jazyků [21]