Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vytvoření knihoven s definovaným rozhraním v jazycích C, C++, C# a FreePascal

Plzeň 2016 Petr Kozler

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. června 2016

Petr Kozler

Abstract

This work examines the possibilities of transfer the source code written in the Java programming language to the C, C++, C# and Pascal programming language. The first part is the research of the availability of libraries with functionality similar to the commonly used Java libraries in those other languages. The next part is implementation of libraries providing Java-like interfaces in other listed languages using the available libraries of these languages, as well as design and implementation of application which allows to search in, view and manipulate their source code. The work should provide a convenient way to translate the code using common Java libraries to the listed languages while keeping its functionality and outline the ways to convert another libraries or imitate the behavior of them in another languages.

Abstrakt

Tato práce se zabývá převodem zdrojových kódů psaných v programovacím jazyku Java do programovacích jazyků C, C++, C# a Pascal. V první části zkoumá dostupnost knihoven s funkcionalitou podobnou základním knihovnám jazyka Java v ostatních uvedených jazycích. Další částí práce je implementace knihoven poskytujících rozhraní v duchu jazyka Java v ostatních uvedených jazycích a využívajících knihovny těchto jazyků a dále návrh a implementace aplikace umožňující vytvořené zdrojové kódy vyhledávat, zobrazovat a dále s nimi manipulovat. Práce by měla umožnit snadný překlad kódu využívajícího základní knihovny jazyka Java do uvedených jazyků se zachováním jeho funkcionality a nastínit způsoby, jakými lze převádět další knihovny do uvedených jazyků nebo napodobit jejich chování v dalších jazycích.

Obsah

1 Úvod

Cílem této práce je usnadnit překlad programového kódu z programovacího jazyka *Java* do jiných programovacích jazyků. Konkrétně se bude jednat o jazyky *C*, *C++*, *C#* a *Pascal* v dialektu překladače Free Pascal. Práce bude sestávat ze čtyř hlavních kroků.

Prvním krokem bude seznámení se s dostupností funkcionality různých běžně používaných metod a tříd z jazyka Java v ostatních výše uvedených jazycích a průzkum toho, jak v těchto jazycích fungují. Předmětem zkoumání bude především paradigma jednotlivých jazyků, dále typový systém, funkce, popř. metody pro matematické výpočty, kontejnery (zejména statická a dynamicky zvětšovaná pole a spojové seznamy), řetězce, funkce nebo metody pro práci s kontejnery (např. řazení a vyhledávání prvků) a funkce, popř. třídy pro vstupní a výstupní operace. Účelem zkoumání bude nalezení nejsnadnějších (s ohledem na efektivitu a přehlednost zdrojového kódu) dostupných způsobů, jak v daných jazycích vytvořit konstrukci funkčně odpovídající danému řešení z jazyka Java.

Druhým krokem bude vytvořit v těchto jazycích knihovny s rozhraním ve stylu jazyka Java, obalující tyto konstrukce. To bude realizováno způsobem, který umožní jejich použití s minimálním počtem úprav zdrojového kódu z jazyka Java, který bude tyto konstrukce využívat, se současným úplným zachováním požadované funkčnosti.

Třetím krokem bude vytvoření aplikace v jazyce Java, která bude sloužit ke správě vytvořených knihoven. Aplikace bude disponovat grafickým uživatelským rozhraním a bude umožňovat jedoduché vyhledávání, zobrazení, přidávání, úpravy a odstraňování jednotlivých knihoven i programovacích jazyků. Data aplikace představující zdrojové kódy knihoven budou ukládána do souborů ve formátu XML. Na chybné vstupy bude aplikace reagovat vhodným hlášením, z něhož bude patrná příčna problému.

Posledním krokem bude otestování celého programového vybavení. Základním prvkem testování budou jednotkové testy, které budou mít za úkol prověřovat, zda se výstupy jednotlivých funkcí nebo metod implementovaných v jazycích C, C++, C# a Pascal shodují s výstupy odpovídajících metod v jazyce Java, a zda chování aplikace pro správu zdrojového kódu odpovídá očekáváním.

2 Analýza metod a funkcí programovacích jazyků

V této kapitole budou popsány charakteristiky jednotlivých jazyků a různé základní funkce a metody, které tyto jazyky poskytují.

2.1 Obecná charakteristika

Základní rozdíly mezi popisovanými jazyky jsou mj. ve způsobu zpracování zdrojového kódu, programovacím paradigmatu, správě paměti a syntaxi.

2.1.1 Java

Java je názvem technologie, jejíž součástí je kromě stejnojmenného vysokoúrovňového (poskytujícího vysokou úroveň abstrakce) *objektově orientovaného* programovacího jazyka i softwarová platforma pro běh programů v tomto jazyce vytvořených. Z hlediska způsobu překladu a spuštění zdrojového kódu se nejedná o čistě kompilovaný ani interpretovaný jazyk, ale o hybridní jazyk. Zpracování programového kódu probíhá následovně [1–4]:

1. Nejprve je proveden překlad veškerého zdrojového kódu (ukládaného do textových souborů, které mají příponu `.java`) s pomocí kompilátoru `javac` do tzv. mezijazyka zvaného *bajtkód*, angl. bytecode (ukládaného v souborech s příponou `.class`). Bytecode je jednodušší formou kódu – jedná se o strojový kód *virtuálního stroje* zvaného Java Virtual Machine (dále jen JVM).

2. Virtuální stroj JVM následně interpretuje (po spuštění nástrojem `java`) kód `.class` souboru do nativního kódu cílového procesoru.

Stejný kód je tedy možné spustit na různých platformách (resp. na platformách, pro které je JVM k dispozici) [1].

Jednou ze základních vlastností jazyka Java je *automatická správa paměti*. Ta umožňuje vytváření objektů bez nutnosti jejich explicitního odstranění za účelem uvolnění paměti ve chvíli, kdy již nejsou potřeba. Objekty, které se nadále nepoužívají (nevedou na ně již žádné odkazy), jsou běhovým prostředím odstraňovány automaticky. Proces automatického likvidování objektů se nazývá garbage collection [1, 2]. Syntax jazyka je zjednodušenou

a mírně upravenou podobou syntaxe jazyků C a C++. Objektový model jazyka je založený na třídách a jazyk podporuje genericitu. S výjimkou základních datových typů jsou všechny proměnné objektem (instancí třídy). Kořenem stromové hierarchie dědičnosti tříd je třída `java.lang.Object` – každá třída je tedy potomkem třídy `java.lang.Object`. Třídy, popř. další konstrukce (rozhraní, výčtové typy) mohou být seskupovány do jmenných prostorů, tzv. balíků (packages) [1, 5].

2.1.2 C

C je nízkoúrovňový, strukturovaný, na funkcích založený jazyk. Pojem nízkoúrovňový znamená, že jazyk přímo pracuje jen se základními datovými typy (znaky, čísla), poskytuje tedy nižší úroveň abstrakce než objektově orientované jazyky jako je např. jazyk Java. Výhodou tohoto přístupu je relativní jednoduchost vytvoření kompilátoru pro konkrétní platformu a efektivita kódu [6, 7].

Jak již bylo nastíněno, na rozdíl od jazyka Java se jedná o jazyk čistě kompilovaný. Kompilace zdrojového kódu jazyka C sestává z několika fází [6, 7]:

1. *Předzpracování* – zpracování kódu tzv. preprocesorem, které spočívá v úpravě kódu do podoby pro snadnější překlad. Součástí této úpravy je např. odstranění komentářů, rozvoj maker, vložení tzv. hlavičkových souborů (soubory s příponu `.h`, obvykle obsahující např. prototypy funkcí, deklarace globálních proměnných nebo definice konstant a globálních typů a obvykle neobsahující výkonný kód) atd.

2. *Překlad* – program vykonávající tuto úlohu se nazývá kompilátor neboli překladač (angl. compiler). Jedná se o hlavní část zpracování zdrojového kódu. Textový soubor kódu upraveného preprocesorem je převeden do objektového neboli relativního kódu cílového počítače (výsledkem práce kompilátoru je soubor `.obj`). Objektový kód představuje téměř hotový spustitelný soubor, pouze adresy proměnných a funkcí jsou zatím zapsány relativně (nejsou zatím známé, protože jsou např. uložené v externí knihovně).

3. *Sestavení* – zpracování relativního kódu vytvořeného překladačem pomocí sestavovacího programu (angl. linker) zahrnuje přidělení absolutních adres a nalezení adres dosud neznámých identifikátorů (jako jsou volané funkce z knihoven). Výstupem této fáze je již hotový program – spustitelný soubor (v OS Windows s příponou `.exe`).

Ze syntaxe tohoto jazyka je odvozena syntaxe všech ostatních jazyků uvedených v této práci s výjimkou jazyka Pascal. Na rozdíl od jazyků Java a C# zde neexistuje mechanismus automatické správy paměti – alokace i dealokace paměti je úkolem programátora. K tomuto účelu nicméně jazyk C poskytuje standardní knihovní funkce. Vzhledem k nutnosti *manuální správy paměti* obsahuje jazyk C na rozdíl od Javy nejen proměnné, obsahující přímo hodnoty, ale též proměnné nazývané jako ukazatele (angl. pointer) na hodnoty daných typů. Ukazatel je proměnná, jejíž hodnotou je adresa v paměti a na této adrese je teprve uložena hodnota (kromě ukazatelů na proměnné existují též ukazatele na funkce). Právě s pomocí ukazatelů jsou v jazyce C realizovány konstrukce podobné složitějším typům jazyka Java, jako jsou řetězce a pole. Objekty jazyka Java lze částečně napodobit s využitím konstrukcí jazyka C pro vytváření nehomogenních (složených z různých datových typů) datových struktur [6, 7].

2.1.3 C++

C++ je jazyk, který je nadstavbou jazyka C – každý platný program v jazyku C je tedy zároveň platným programem v jazyku C++. Princip zpracování zdrojového kódu je stejný jako v případě jazyka C (některé překladače fungují, nebo dříve fungovaly tak, že nejprve převedli zdrojový kód C++ na zdrojový kód C, který byl poté předán překladači jazyka C) [8].

Stejně jako jazyk C i jazyk C++ vyžaduje manuální správu paměti. Pro tento účel jsou nicméně k dispozici nové jazykové prostředky, např. konstruktory a destruktory nebo reference na proměnné (jako alternativa k ukazatelům). Kromě toho obsahuje jazyk C++ řadu dalších nových konstrukcí, jednou z nejdůležitějších je třída. Rozdílem oproti C je tedy mj. podpora objektově orientovaného programování (vedle procedurálního přístupu z jazyka C) a generického programování (prostřednictvím tzv. šablon). Jazyk C++ lze proto označit jako multiparadigmatický [8, 9].

Mezi další nové konstrukce patří jmenné prostory (obdoba balíků v jazyku Java), přetěžování funkcí i operátorů, standardní parametry, výše zmíněné šablony, vložené (angl. inline) funkce a další. Kromě nových konstrukcí přináší C++ oproti C také rozsáhlou sadu nových knihoven, přičemž je stále možné používat i knihovny z jazyka C [8].

2.1.4 C#

C# je vysokoúrovňový, objektově orientovaný jazyk vycházející z jazyků, jako je C++ a Microsoft Visual Basic a je zaměřený především na vývoj

aplikací pro platformu .NET Framework [10, 11].

Způsob zpracování programového kódu je postaven na podobných principech jako v případě jazyka Java. Obdobou JVM je *Common Language Runtime* a obdobou bytecode je *Microsoft Intermediate Language*. Syntax je velmi podobná jazyku Java, navíc ale obsahuje konstrukce jako jsou například částečné třídy (třídy, jejichž definice je rozdělena do více zdrojových souborů), delegáty, přetěžování operátorů (nejen metod), předávání parametrů metod referencí, struktury (třídy, jejichž instance jsou vytvářené na zásobníku) nebo bezznaménkové číselné datové typy [10, 12–15].

2.1.5 Pascal

Pascal je, stejně jako C a C++, kompilovaný jazyk, pro který je k dispozici řada překladačů (každý z nich definuje mírně odlišnou podobu jazyka, tzv. dialekt). Jedním z nich je i volně šiřitelný překladač *Free Pascal*, který je použit v rámci této práce. Všechny poznámky týkající se Pascalu se tedy vztahují na dialekt překladače Free Pascal a nemusejí platit pro ostatní překladače [16].

Vedle způsobu zpracování kódu je Pascal srovnatelný s jazykem C++ také po stránce správy paměti a v podporovaných programovacích paradigmatech – je tedy podporováno jak strukturované programování založené na procedurách a funkcích (jako procedury jsou nazývány podprogramy bez návratové hodnoty), tak objektově orientované programování [16].

Na první pohled odlišná je syntax jazyka, která je méně úsporná a je přizpůsobena pro výuku. Soubor zdrojového kódu programu se nazývá modul nebo jednotka (angl. unit), obvyklá přípona souboru je `.pas` a zdrojový kód má pevně danou strukturu. Jeho základem je blok, který se skládá z hlavičky, deklarace a příkazové části. Deklarační část může obsahovat podprogramy, které mají stejnou strukturu [16, 17].

2.2 Základní datové typy

Všechny uvedené jazyky jsou typované – při deklaraci proměnné je nutné uvést její datový typ. Každý z těchto jazyků poskytuje množinu základních datových typů. Jejich seznam spolu s uvedenými rozsahy hodnot je uveden níže.

Tabulka 2.1: Základní datové typy jazyka Java

Název	**Význam**	**Rozsah**
`byte`	celé číslo	-2^7 .. $2^7 - 1$
`short`	celé číslo	-2^{15} .. $2^{15} - 1$
`int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long`	celé číslo	-2^{63} .. $2^{63} - 1$
`float`	reálné číslo	dle IEEE 754 (jednoduchá přesnost)
`double`	reálné číslo	dle IEEE 754 (dvojitá přesnost)
`boolean`	logická hodnota	`true` nebo `false`
`char`	znak Unicode	`'\u0000'` (0) .. `'\uffff'` ($2^{16} - 1$)

2.2.1 Java

Jazyk Java je (stejně jako ostatní uvedené jazyky) *staticky typovaný*, což znamená, že každou proměnnou je nutné před použitím deklarovat a kromě jména je nutné uvést její datový typ, který určuje množinu povolených hodnot a operací nad nimi. K dispozici je 8 primitivních datových typů (typů předdefinovaných jazykem, jejichž název je klíčovým slovem jazyka), z toho 4 celočíselné, 2 pro reálná čísla, 1 znakový a 1 pro logické hodnoty. Všechny číselné datové typy jsou znaménkové [1, 18].

Jednotlivé datové typy, jejich význam, velikost v paměti a rozsahy hodnot popisuje tabulka 2.1. Pro uvedené typy jsou k dispozici tzv. *obalové třídy*, umožňující další operace (mj. slouží k uchovávání hodnot primitivních datových typů jako objektů pro použití v kolekcích); např. `java.lang.Integer` nebo `java.lang.Long`, které např. umožňují použití hodnot číselných datových typů jako bezznaménkových. Kromě těchto typů poskytuje jazyk Java speciální podporu pro řetězce reprezentované třídou `java.lang.String` (popsány v kapitole 2.4) [18–26].

2.2.2 C

Primitivní datové typy jazyka C zahrnují typy celočíselné a typy pro reálná čísla (vestavěný typ pro logické hodnoty jazyk C neobsahuje, místo něj lze logické hodnoty reprezentovat pomocí hodnot celočíselného typu). Velikosti a rozsahy jednotlivých datových typů závisí na cílové platformě. Garantovány jsou pouze minimální rozsahy a pořadí velikosti rozsahů pro jednotlivé datové typy (není však vyžadováno, aby jednotlivé datové typy měly rozsahy rozdílné velikosti). Definice celočíselných datových typů s garantovanými rozsahy hodnot obsahuje soubor `stdint.h`, přičemž některé z nich jsou volitelné a jejich podpora proto není zaručena na všech platformách [7, 27].

Tabulka 2.2: Některé základní datové typy jazyka C

Název	**Význam**	**Rozsah**
`char`	celé číslo	-2^7 .. $2^7 - 1$
`short int`	celé číslo	-2^{15} .. $2^{15} - 1$
`int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long long int`	celé číslo	-2^{63} .. $2^{63} - 1$
`float`	reálné číslo	`FLT_MIN` .. `FLT_MAX`
`double`	reálné číslo	`DBL_MIN` .. `DBL_MAX`
`long double`	reálné číslo	`LDBL_MIN` .. `LDBL_MAX`

Minimální rozsahy základních typů jsou uvedené v tabulce 2.2. Typy jsou seřazeny vzestupně podle minimální velikosti. Kromě uvedených datových typů existuje ke každému celočíselnému typu bezznaménková varianta – její název začíná řetězcem `unsigned`. Takový typ má stejnou velikost jako příslušný znaménkový typ a jeho rozsah je posunutý tak, že nejmenší hodnota je 0 (tedy pouze kladná čísla). Celočíselné datové typy `long long int` a `unsigned long long int` nejsou součástí standardu C89. Konstanty představující hraniční hodnoty datových typů pro reálná čísla jsou uloženy v hlavičkovém souboru `float.h` a platí pro ně, že typ `float` má nejmenší rozsah a zbylé dva typy mají rozsah minimálně tak velký, ale obvykle větší, než `float`. Konstanty plnící stejný účel pro celočíselné typy definuje soubor `limits.h` [7].

2.2.3 C++

Jak již bylo uvedeno v kapitole 2.1.3, C++ je nadstavbou jazyka C, proto každý datový typ, který obsahuje jazyk C, obsahuje i C++, a platí pro něj stejná pravidla, jako v jazyku C. Jazyk C++ nicméně obsahuje nové fundamentální datové typy navíc, příkladem je datový typ pro logické hodnoty `bool`, který je obdobou typu `boolean` v jazyku Java [8, 28].

2.2.4 C#

Na rozdíl od jazyka Java jsou v jazyku C# všechna klíčová slova, představující jednoduché vestavěné datové typy ve skutečnosti *aliasy* pro třídy předdefinované ve jmenném prostoru System. Ty zde kromě znakového, logického a číselných datových typů (které jsou podobné těm v jazyku Java a rovněž mají garantované velikosti a rozsahy) zahrnují i řetězce (popsány

Tabulka 2.3: Některé základní datové typy jazyka C#

Název	**Význam**	**Rozsah**
`sbyte`	celé číslo	-2^7 .. $2^7 - 1$
`short`	celé číslo	-2^{15} .. $2^{15} - 1$
`int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long`	celé číslo	-2^{63} .. $2^{63} - 1$
`float`	reálné číslo	dle IEEE 754 (jednoduchá přesnost)
`double`	reálné číslo	dle IEEE 754 (dvojitá přesnost)
`bool`	logická hodnota	`true` nebo `false`
`char`	znak Unicode	`'\u0000'` (0) .. `'\uffff'` ($2^{16} - 1$)

v kapitole 2.4) a obecné objekty [10, 15, 29–36].

Příklady předdefinovaných typů jsou popsány v tabulce 2.3. Dále ke každému číselnému datovému typu existují i bezznaménkové varianty stejné velikosti `byte`, `ushort`, `uint` a `ulong`. Pro popis reálných čísel je mimo typů uvedených v tabulce k dispozici typ `decimal`, který má menší rozsah, ale větší přesnost. Pro řetězec Unicode znaků je v C# k dispozici předdefinovaný typ `string` a pro obecný objekt typ `object` [10, 15, 37–43].

2.2.5 Pascal

Typový systém jazyka Pascal obsahuje celou řadu základních datových typů (větší množství než ostatní uvedené jazyky), které se člení na typy základní (ty se dále dělí na ordinální, zahrnující mj. typy celočíselné a logické, a reálné), znakové a další [44–46].

Pro účely této práce zde budou v tabulce 2.4 popsány pouze typy, které lze použít jako obdobu primitivních datových typů jazyka Java. Vynechány budou číselné a logické typy jiných rozsahů, než jaké poskytují základní typy v jazyce Java (například bezznaménkové celočíselné typy `Byte`, `Word`, `Longword` a `QWord` nebo některé typy pro reálná čísla, např. `Currency`) a stejně tak budou vynechány typy, jejichž velikost je platformově závislá a pro použití jako obdoby typů jazyka Java jsou tedy méně vhodné, než jejich alternativy s pevnou velikostí (zejména `Integer` a `Real`) [44, 47–49].

Typ `Char` zde slouží jako synonymum pro typ `AnsiChar`, který má velikost 1 byte. Vedle něho existuje v jazyku Pascal i typ `WideChar` s velikostí 2 byte, který reprezentuje Unicode znak [44, 50, 51].

Problematika znaků a řetězců bude podrobněji rozebrána v kapitole 2.4.

Tabulka 2.4: Některé základní datové typy jazyka Pascal

Název	**Význam**	**Rozsah**
`Shortint`	celé číslo	-2^7 .. $2^7 - 1$
`Smallint`	celé číslo	-2^{15} .. $2^{15} - 1$
`Longint`	celé číslo	-2^{31} .. $2^{31} - 1$
`Int64`	celé číslo	-2^{63} .. $2^{63} - 1$
`Single`	reálné číslo	dle IEEE 754 (jednoduchá přesnost)
`Double`	reálné číslo	dle IEEE 754 (dvojitá přesnost)
`Boolean`	logická hodnota	`True` nebo `False`
`Char`	znak	`'#0'` (0) .. `'#255'` ($2^8 - 1$)

2.3 Pole a seznamy

Pro uchovávání většího množství hodnot bez nutnosti vytváření proměnné pro každou proměnnou zvlášť poskytují uvedené programovací jazyky datovou strukturu pole, případně složitější datové struktury, jako jsou dynamicky alokované seznamy.

2.3.1 Java

Pole v jazyku Java je objekt, který uchovává předem určené množství hodnot jednoho (libovolného) typu. Délka daného pole (počet prvků) je určena v průběhu jeho vytvoření a dále se nemění. Číslování prvků začíná od indexu 0 [52].

Pro pokročilejší manipulaci s poli je k dispozici knihovní třída `java.util.Arrays`. Tato třída poskytuje metody pro kopírování, porovnávání či naplnění polí hodnot různých typů, jejich převod do řetězcové reprezentace, řazení nebo binární vyhledávání prvků [53].

Jazyk Java dále poskytuje třídy implementující pokročilejší dynamické datové struktury pro uchovávání prvků (také nazývané jako kolekce). Tyto třídy implementují rozhraní `java.util.Collection<E>` a zahrnují např. polem implementovaný seznam (třída `java.util.ArrayList<E>`), spojový seznam (třída `java.util.LinkedList<E>`) nebo množiny (např. `java.util.HashSet<E>`). Mezi další datové struktury, které jsou k dispozici, patří např. mapy (`java.util.HashMap<K, V>`) [54–58].

Pro účely této práce jsou důležité především seznamy. Třídy `java.util.ArrayList<E>` a `java.util.LinkedList<E>` implementují rozhraní `java.util.List<E>`, mezi jehož metody patří metody pro vkládání, úpravu, odstraňování a výběr prvků, zjišťování aktuálního počtu prvků nebo převod

prvků seznamu do textové reprezentace [55, 56, 59].

Další manipulaci se seznamy (případně jinými kolekcemi) umožňuje třída `java.util.Collections`, která je obdobou třídy `java.util.Arrays`. Mezi poskytované metody tak patří kopírování, řazení, vyhledávání a další [60].

Seznamy `java.util.ArrayList<E>` a `java.util.LinkedList<E>` jsou generické, stejně jako další kolekce, metody třídy `java.util.Collections` a některé metody třídy `java.util.Arrays` [53, 55, 56, 60].

Řazení polí a seznamů, vyhledávání jejich prvků a porovnávání prvků mezi sebou se bude podrobněji věnovat kapitola 2.5.

2.3.2 C

V jazyku C je pole definováno jako posloupnost prvků stejného typu uložených za sebou v paměti. Velikost pole je určena při deklaraci. Stejně jako v jazyku Java se zde indexy prvků číslují od 0 [61].

Pole zde úzce souvisí s ukazateli. Ukazatel uchovává adresu v paměti pro danou proměnnou. Při uložení více prvků stejné hodnoty za sebou v paměti je hodnota na adrese, kam směřuje ukazatel, totéž jako první prvek odpovídajícího pole; přičtení určitého čísla k adrese ukazatele (s využitím ukazatelové aritmetiky) je pak totéž jako přistoupení k prvku pole na odpovídajícím indexu. Pole je vždy možné zkonvertovat na ukazatel příslušného typu [62].

Pro realizaci dynamicky zvětšovaných datových struktur lze použít knihovní funkce pro alokaci paměti za běhu programu. Příkladem jsou funkce `malloc()` a `calloc()` definované v hlavičkovém souboru `stdlib.h`, které vrací ukazatel na začátek souvislého bloku nově alokované paměti [63, 64].

2.3.3 C++

Pro obyčejná pole zde platí stejná pravidla jako pro pole v jazyku C. Pro pokročilejší datové struktury jsou nicméně k dispozici generické knihovní třídy jako `std::vector`, `std::list`, `std::set`, `std::map` a další [65–68].

Dynamické pole představuje třída `std::vector`, spojový seznam pak třída `std::list`. Jejich členské funkce (metody) pak poskytují podobnou funkcionalitu jako jejich protějšky `ArrayList` a `LinkedList` v jazyku Java [65, 66].

2.3.4 C#

Úlohu pole v jazyku C# plní třída `System.Array`. Tato třída zároveň poskytuje funkcionalitu srovnatelnou s třídou `Arrays` jazyka Java, tedy metody

pro řazení, vyhledávání, kopírování atd. Indexy se číslují stejně jako v jazyce Java [69, 70].

Obdobu `ArrayListu` jazyka Java zde představuje generická třída `System.Collections.Generic.List<T>` a pro spojový seznam je k dispozici generická třída `System.Collections.Generic.LinkedList<T>`. Obdobně jako v případě třídy `System.Array`, i tyto třídy mají metody jak pro základní operace (přidávání prvků, výběr apod.), tak pro pokročilé operace v duchu třídy `Collections` z jazyka Java [71, 72].

2.3.5 Pascal

Pole se v jazyce Pascal řadí mezi tzv. strukturované typy (typy, které uchovávají více hodnot pomocí jedné proměnné). Pascal obsahuje jak statická (velikost, resp. počáteční a konečný index je určen při deklaraci), tak dynamická (velikost je určena za běhu programu funkcí `SetLength()`) pole. U statických polí závisí číslování indexů prvků na deklaraci, u dynamických se čísluje od 0. Kromě statických a dynamických polí Pascal, stejně jako C, podporuje ukazatele [73–76].

Seznam s proměnlivou velikostí je v Pascalu podporován prostřednictvím několika tříd, jako je `TList`, `TFPList` a generický `TFPGList` a `TFPGObjectList`, s rozhraním podobným rozhraní `List` z Javy [77–80].

2.4 Textové řetězce

Uvedené jazyky obsahují rozdílné datové struktury, příp. datové typy pro práci s textem. Dále poskytují různé funkce nebo metody pro základní operace nad těmito strukturami.

2.4.1 Java

Řetězce znaků reprezentují instance třídy `java.lang.String`. Ty jsou neměnné – po vytvoření řetězce nelze změnit jeho hodnotu, pouze vytvořit nový (pro větší množství úprav jednoho textu jsou k dispozici třídy jako `java.lang.StringBuffer` a `java.lang.StringBuilder`). Kromě řetězcového literálu lze objekt `java.lang.String` vytvořit např. pomocí pole bajtů nebo znaků. Řetězce lze spojovat (tzv. konkatenace) pomocí operátoru pro sčítání „+“ [81–84].

Třída `java.lang.String` má metody pro různé operace nad řetězci. Patří mezi ně metody pro porovnávání řetězců, určování délky, hledání nebo

vytváření podřetězců, nahrazování znaků, převody na velká nebo malá písmena, odstraňování bílých znaků (trimování) a v neposlední řadě např. rozdělování řetězců do polí podřetězců podle určené množiny oddělovačů [82].

K rozdělování řetězce do podřetězců (tokenů) podle oddělovacích znaků (delimiterů) je možné použít i třídu `java.util.StringTokenizer`. Při vytváření instance je určen řetězec k rozdělení a řetězec oddělovačů (každý jednotlivý znak tohoto řetězce je považován za oddělovač). Posléze jsou postupně načítány jednotlivé tokeny (v průběhu načítání lze zjišťovat jejich počet) [85].

Metody pro získání hodnoty jiného (např. číselného) datového typu z její řetězcové reprezentace poskytují obalové třídy příslušných typů [19–26, 82].

2.4.2 C

Pro reprezentaci řetězců se obvykle používá pole hodnot typu `char`. Řetězec v C je tedy polem znaků, které je zakončené ukončovací značkou, tzv. terminátorem neboli nulovým znakem (má hodnotu 0) [86].

Funkce pro manipulaci s řetězci definuje hlavičkový soubor `string.h`. Jedná se např. o funkce pro porovnávání (`strcmp()`), kopírování (`strcpy()`) a konkatenaci (`strcat()`) řetězců, zjišťování délky řetězce (`strlen()`), a také načítání tokenů z řetězce podle určených oddělovačů (`strtok()`) v duchu třídy `StringTokenizer` z Javy [87].

Soubor `stdlib.h` pak definuje mj. funkce pro získání hodnot číselných typů z jejich řetězcové reprezentace – `strtol()`, `strtod()` apod [88].

Zásadní odlišností znakového typu `char` (a tím i řetězci) v jazyku Java (a C#) oproti typu char jazyka C je jeho velikost – Java používá 16-bitové *Unicode* znaky, zatímco typ `char` v C je pouze 8-bitový [28].

Pro účely této práce je nicméně možné se omezit na 8-bitové kódování. Pro jednoduchost implementace knihoven v rámci realizační části této práce tak budou v jazycích C, C++ a Pascal používány pouze základní 8-bitové znaky a odpovídající implementace řetězců.

2.4.3 C++

C++ oproti C obsahuje zlepšenou podporu řetězců prostřednictvím třídy `std::string` definované v hlavičkovém souboru `string` standardní knihovny jazyka C++. Tato třída poskytuje některé metody se stejnou funkcionalitou jako metody třídy `String` z Javy. Prostřednictvím přetížení operátoru sčítání taktéž umožňuje konkatenaci řetězcových literálů ve stylu jazyka Java. Dále umožňuje získání ekvivalentního pole znaků reprezentujícího řetězec ve

stylu jazyka C nebo převod textové reprezentace číselné hodnoty na odpovídající číselnou hodnotu daného typu [89, 90].

2.4.4 C#

Třída `System.String` je obdobou stejnojmenné třídy jazyka Java. Zde je možné použít alias `string`. Vnitřně je text uložen jako posloupnost Unicode znaků. Pro objekty `string` zde platí obdobná pravidla jako pro objekty `String` v Javě, včetně funkcionality poskytovaných konstruktorů a metod. Navíc je ale možné například přistupovat k jednotlivým znakům řetězce stejným způsobem, jako k prvkům pole [91].

K rozdělování řetězce na tokeny se zde používá instanční metoda `Split()`. Tato metoda vrací pole vzniklých podřetězců rozdělených podle předaných oddělovačů. Jednotlivé oddělovače zde mohou být nejen znaky, ale i celé řetězce (jsou tedy předány jako pole řetězců, nikoliv jako jediný řetězec, jehož znaky jsou jednotlivé oddělovače) [92].

2.4.5 Pascal

Řetězce jako posloupnosti 8-bitových znaků představuje typ `AnsiString`. Pro takové řetězce existuje i typ `ShortString`, ten má ale omezenou délku a jeho použití pro účely této práce není vhodné. Vnitřně je řetězec typu `AnsiString` realizován jako ukazatel a je zakončen nulovým znakem; ten však neslouží pro určení délky řetězce, nulové znaky je tak možné použít i jako součást řetězce [93–95].

Pro manipulaci s řetězci jsou poskytnuty různé funkce, např. `Length()` pro určení délky, `Copy()` pro kopírování, `Pos()` pro nalezení znaku, dále `Trim()`, `UpperCase()`, `LowerCase()` a řada dalších. K rozdělení řetězce na tokeny je možné využít třídy `TStringList`, resp. `TStrings` [96–98].

2.5 Řazení a vyhledávání

V uvedených programovacích jazycích jsou dostupné funkce nebo metody pro řazení prvků v polích a další operace s poli. Tyto operace vyžadují v jednotlivých jazycích rozdílné způsoby porovnávání prvků.

2.5.1 Java

Řazení a binární vyhledávání prvků v poli nebo seznamu zajišťují metody `sort()`, resp. `binarySearch()`, kterými disponují knihovní třídy `java.util-`

`.Arrays`, resp. `java.util.Collections` [53, 60].

Pro seřazení pole primitivních datových typů je implementován algoritmus *Quicksort*. Pro řazení pole nebo seznamu objektů je použit stabilní řadící algoritmus *Mergesort*. Prvky jsou standardně řazeny vzestupně [53, 60].

Pro všechny řazené prvky musí platit, že implementují rozhraní `java.lang.Comparable<T>` a jsou vzájemně porovnatelné. Alternativou je řazení objektů s využitím komparátoru (třída implementující rozhraní `java.util.Comparator<T>`). Stejné pravidlo pro porovnávání pak platí v případě binárního vyhledávání [53, 60, 99, 100].

2.5.2 C

Pro účely řazení a binárního vyhledávání prvků definuje hlavičkový soubor `stdlib.h` funkce `qsort()` (která implementuje algoritmus Quicksort) a `bsearch()`. Obě funkce přijímají ukazatel na obecné pole, kromě počtu jeho prvků je tak nutné dále uvést velikost jednoho prvku v bajtech a dále ukazatel na funkci pro vzájemné porovnání dvou prvků tohoto pole [101, 102].

2.5.3 C++

Hlavičkový soubor `algorithm` definuje celou řadu šablon funkcí pro řazení a vyhledávání prvků v různých posloupnostech prvků včetně polí a seznamů. Pro kompletní vzestupné řazení prvků v definovaném rozsahu je určena šablona funkce `std::sort()`, resp. `std::stable_sort()` v případě požadavku na stabilní řazení. Pro binární vyhledávání je určena šablona funkce `std::binary_search()` [103–105].

Všechny uvedené funkce kromě iterátoru na první a poslední prvek posloupnosti přejímají též ukazatel na funkci, nebo funkční objekt (instance třídy, která má jako veřejnou instanční metodu operátor volání funkce) pro vzájemné porování dvou prvků [103–105].

2.5.4 C#

Třída `System.Array`, která reprezentuje pole, a třída `System.Collections.Generic.List<T>`, která představuje generický seznam implementovaný polem proměnlivé délky, poskytují mj. metody pro řazení a vyhledávání prvků [70, 71].

Stejně jako v případě jazyka Java, i zde je několik možností, jak při řazení nebo vyhledávání porovnávat objekty. Jednou z možností je implementace rozhraní `System.IComparable<T>`. Další možností je poskytnutí příslušného

komparátoru – objektu, jenž implementuje rozhraní `System.Collections-.Generic.IComparer<T>`. Navíc je zde možné využití delegátů – namísto porovnávací třídy se v tomto případě poskytne odkaz na porovnávací metodu `System.Comparison<T>` [106–108].

2.5.5 Pascal

Pro třídy implementující seznamy (jako `TStringList`, `TList`, `TFPList` a `TFPGList`) jsou k dispozici metody pro řazení prvků algoritmem Quicksort, v případě některých (`TStringList`) též pro binární vyhledávání [109, 110].

Pro porovnávání prvků přijímají dotyčné metody jako parametr porovnávací funkci [111].

2.6 Matematické výpočty

Každý uvedený jazyk vedle nejzákladnějších aritmetických operátorů disponuje knihovnou poskytující běžné matematické funkce např. pro určení absolutní hodnoty, nalezení menšího nebo většího prvku nebo výpočet odmocniny.

2.6.1 Java

Metody představující běžné matematické funkce jsou k dispozici v knihovní třídě `java.lang.Math`. Mezi podporované matematické funkce patří absolutní hodnota, mocniny a odmocniny, goniometrické a cyklometrické funkce, logaritmy, hledání minima a maxima z dvojice čísel nebo zaokrouhlování. Dále obsahuje definice běžných matematických konstant, čísla π a čísla e [112].

2.6.2 C

Funkce pro určení absolutní hodnoty (např. `abs`) a celočíselné dělení (např. `div`) jsou definovány v souboru `stdlib.h`. Funkce pro ostatní matematické výpočty v souboru `math.h` a zahrnují mj. většinu funkcí uvedených u třídy `Math` v jazyku Java. Příklady: funkce `pow()`, `sqrt()`, `sin()`, `cos()`, `tan()`, `log()`, `log10()`, `min()`, `max()`, `round()` [88, 113].

2.6.3 C++

Soubor `cmath` pro C++ definuje dodatečné knihovní funkce pro matematické výpočty, jako je například funkce `::abs()` pro výpočet absolutní hodnoty

přetížená pro různé celočíselné typy [88, 114].

2.6.4 C#

Obdobně jako v jazyku Java, i zde existuje knihovní třída `System.Math`, která plní stejnou úlohu. Množina dostupných matematických funkcí a konstant je obdobná jako v případě knihovny `Math` v jazyku Java [115].

2.6.5 Pascal

Některé matematické funkce (vedle celé řady jiných), jako např. `Abs()`, `Sin()`, `Cos()`, `Tan()`, `Power()`, `Sqr()`, `Sqrt()` nebo `Round()` jsou definovány v modulu `System`, další jsou pak pokryty dodatečně v modulu `Math` (příklady: `Min()`, `Max()`, `Log10()`, `RoundTo()`) [96, 116].

2.7 Vstup a výstup

Součástí uvedených jazyků jsou knihovny pro vstupní a výstupní operace pro standardní vstup a výstup, pro soubory, sockety atd.

2.7.1 Java

Pro načítání uživatelského vstupu je v jazyce Java k dispozici několik tříd, jednou z nich je třída `java.util.Scanner`. Tato třída načítá vstup ze zdroje předaného v konstruktoru, kterým může být např. soubor, již existující textový řetězec nebo instance třídy `java.io.InputStream`, kterou může být např. standardní vstup (výchozí objekt odkazovaný statickým atributem `in` knihovní třídy `java.lang.System`) nebo vstupní proud (angl. stream) ze socketu [117–119].

Třída `java.util.Scanner` obsahuje metody jak pro čtení řádek, tak pro čtení jednotlivých tokenů buď jako řetězců, nebo jako hodnot různých jiných datových typů, např. čísel. Tokeny načtené jako řetězce je taktéž možné dodatečně převést na hodnoty jiných typů pomocí metod s názvem začínajícím na řetězec `parse-` obalových tříd příslušných datových typů (metody `toString()` těchto tříd provádějí opačnou operaci) [19–26, 117].

Pro výstup je rovněž dostupných několik tříd. Obecný výstupní proud (např. do souboru, do socketu, na obrazovku) představuje třída `java.io.OutputStream`. Pohodlnější způsob zapisování představuje např. jeden z jejích (nepřímých) potomků, třída `java.io.PrintStream`, která je typem statických atributů `out` a `err` (výchozí objekty představují standardní a

chybový výstup) knihovní třídy `java.lang.System`. Její metody umožňují kromě zápisu řetězců i zápis primitivních datových typů nebo jiných objektů (voláním jejich metody `toString()`), obojí s možností zakončení znakem nového řádku. Dále poskytuje i metodu pro výpis řetězce naformátovaného z předaného formátovacího řetězce a objektů představujících jeho argumenty. V tomto případě se vypíše návratová hodnota metody `toString()` těchto objektů [119–121].

2.7.2 C

Vstupní a výstupní operace jsou prováděny pomocí funkcí definovaných v hlavičkovém souboru `stdio.h`. Ke čtení ze standardního vstupu, jakož i k zápisu na standardní výstup existuje několik funkcí. Funkce `gets()` a `getchar()` umožňují jednoduché čtení po řádcích, resp. po jednotlivých znacích. Funkce `puts()` a `putchar()` umožňují jednoduchý zápis po řádcích, resp. po jednotlivých znacích [122].

Pro formátovaný vstup je k dispozici funkce `scanf()`, která ze standardního vstupu načítá hodnoty podle formátovacího řetězce předaného jako první parametr funkce a ukládá je do proměnných, jejichž adresy přebírá jako další parametry funkce. Pro formátovaný výstup je k dispozici funkce `printf()`, která analogickým způsobem vypisuje formátovaný řetězec, kdy prvním parametrem je řetězec ke zformátování a dalšími parametry jsou proměnné, jejichž hodnoty jsou do řetězce dosazeny [123, 124].

2.7.3 C++

Standardní vstup a výstup představuje objekt `std::cin`, resp. `std::cout`. Oba jsou definované v souboru `iostream`. První jmenovaný je instancí třídy `istream`, která používá operátor „>>“ pro načtení znaků jako formátovaných dat do požadované proměnné. Jako alternativa existuje metoda `read()`. Druhý jmenovaný objekt je instancí třídy `ostream`, ta používá operátor „<<“ pro zápis znaků jako formátovaných dat. Alternativou je metoda `write()` [125–127].

2.7.4 C#

Standardní vstupní, výstupní a chybový výstupní proud představuje knihovní třída `System.Console` [128].

Její statické metody pro čtení umožňují načítání uživatelského vstupu po řádcích i po jednotlivých znacích. Takto načtený řetězec nebo znak lze

převést na jiný základní datový typ (nebo obráceně) pomocí metod knihovní třídy `System.Convert`. Řetězec lze také převést na hodnotu jiného datového typu pomocí metody `Parse()` nebo `TryParse()` příslušného datového typu [29–36, 129].

Statické metody třídy `System.Console` pro zápis umožňují zapisování jak řetězců, tak jiných datových typů (prostřednictvím voláním metody `ToString()`), buď za sebou, nebo po řádcích. Umožňují i formátovaný zápis v duchu odpovídající metody třídy `PrintStream` v jazyku Java [128].

2.7.5 Pascal

Modul `System` definuje základní procedury pro vstupní a výstupní operace. Jsou jimi procedury `Read()`, `ReadLn()`, `Write()` a `WriteLn()`. Procedura `Read()` slouží k načítání hodnot do proměnných předaných jako argumenty. Procedura `Write()` slouží k zápisu hodnot proměnných předaných jako argumenty. Varianty s názvem končícím na řetězec `-Ln` slouží ke čtení po řádcích, resp. zápisu s ukončením řádky [96, 130–133].

Procedury mohou mít jeden nebo dva parametry. V prvním případě parametr představuje proměnnou, do které má být vstup načten či která má být vypsána na výstup. V druhém případě je první parametr odkaz na soubor, se kterým se má pracovat místo standardního vstupu nebo výstupu [130–133].

3 Implementace knihoven

Tato kapitola popisuje způsoby, jakými byly implementovány jednotlivé typy knihoven s rozhraním v duchu jazyka Java do ostatních popisovaných jazyků.

3.1 Pravidla při implementaci

Při implementaci knihoven byla (v rámci možností každého z jazyků) za účelem zpřehlednění kódu a maximální podobnosti s knihovnami jazyka Java dodržována následující pravidla:

1. Názvy funkcí, resp. metod, které tvoří rozhraní knihoven, jakož i příp. názvy tříd, jsou pojmenovány stejnými názvy, jako jejich předlohy v jazyce Java, s výjimkou případů, kdy to daný jazyk neumožňuje (např. pokud takový název je v daném jazyce klíčovým slovem nebo názvem některé ze základních součástí vestavěných knihoven jazyka a není možné jej překrýt).

2. Pokud jazyk neumožňuje použití názvu některé třídy nebo metody z jazyka Java, je tento název pozměněn v nejmenší možné míře, aby byl umožněn triviální přepis z konstrukce v jednom jazyce do konstrukce v jiném. Jako nepříliš běžný, avšak jednoduchý způsob umožňovaný všemi jazyky se ukázalo zejm. doplnění znaku podtržítka „_“ za příslušný název.

3. Metody tvořící rozhraní tříd jsou označeny příslušným modifikátorem jako veřejné. To neplatí pro jazyk C, který třídy nepodporuje.

4. Funkce v jazyku C, které nejsou určeny jako součást rozhraní knihoven, nejsou uváděny v příslušných hlavičkových souborech a jejich název začíná vždy znakem podtržítka (narozdíl od částí rozhraní, které kolidují s konstrukcemi daného jazyka a které podtržítkem končí). Stejným způsobem jsou nazývány i metody v případě jiných jazyků, které nejsou určeny jako součást rozhraní (odpovídající knihovna jazyka Java je neobsahuje), ale bylo nutné je neoznačovat jako privátní (např. jsou využívány vícero knihovnami).

5. Není-li pro konkrétní množinu případů uvedeno jinak, pro identifikátory v kódu netvořícím rozhraní byl ve všech jazycích použit stejný

styl jako v případě kódu, který rozhraní představuje, a který odpovídá konvencím jazyka Java, tedy *camelCase*.

6. Není-li daným jazykem umožněno přetěžování funkcí (případ jazyka C), pak jsou názvy funkcí odpovídajících přetíženým metodám jednoduchým způsobem obměňovány (např. doplněním zkratky představující název datového typu parametrů funkce za název této funkce nebo doplněním slova vystihujícího rozdíl oproti jiné obměně této funkce), aby se předešlo kolizím názvů.

7. Oproti pravidlům pro pojmenování samotných identifikátorů, které se snaží dodržet maximální textovou podobnost s jazykem Java, v případě celkového návrhu knihoven je dodržována přednost funkční podobnosti před podobností textovou. Znamená to, že např. pro práci s objekty v jazyce C++ je využíváno ukazatelů (podobně, jako proměnné objektového typu v Javě představují odkazy na objekty) s tím, že k atributům a metodám těchto objektů je přistupováno pomocí dvojice znaků „->“, přestože textově podobnější jazyku Java by v tomto případě byla přímá práce s hodnotou objektové proměnné s přístupem k metodám pomocí znaku tečky „.“.

8. Funkce v jazycích C a C++, které jako parametry přebírají vestavěná pole, obvykle zároveň přebírají jako další parametr bezprostředně za každým jednotlivým polem celé číslo, které představuje délku tohoto pole. To neplatí, pokud jsou hraniční indexy v poli určeny jiným způsobem (zejména z parametrů, které představují počáteční a konečný index přístupu do pole).

9. Funkce v jazyce C, které odpovídají instančním metodám Java třídy, přebírají vždy jako první parametr ukazatel na alokovanou stavovou strukturu příslušného typu.

10. Funkce v jazyce C, které odpovídají statickým metodám Java třídy, globální proměnné odpovídající statickým atributům a identifikátory hodnot představující veřejné konstanty definované pomocí maker preprocesoru, mají ve svém názvu vždy prefix v podobě názvu dotyčné třídy oddělený od zbytku názvu podtržítkem, čímž se předchází kolizím názvů napříč vytvořenými moduly.

3.2 Základní třídy pro operace s objekty

S výjimkou jazyka C, který neumožňuje objektový přístup, byla ve všech jazycích vytvořena třída `Object` s implementovanou metodou `equals()` pro testování shody objektů a metodou `toString()` pro výpis textové reprezentace objektů. Po vzoru jazyka Java každá následující uvedená vytvořená třída (s výjimkou tříd, ze kterých nejsou vytvářeny instance a poskytují pouze statické metody) dědí od této třídy, což umožňuje snadnější práci s objekty v některých knihovnách.

Kromě třídy `Object` byly imitovány rozhraní `Comparable` a `Comparator` pro porovnávání objektů za účelem řazení a vyhledávání, a to ve všech jazycích v obou případech jako třídy dědící od třídy `Object` (přesněji abstraktní třídy v případě jazyka C#, třídy obsahující pouze abstraktní metody v případě Pascalu a třídy obsahující pouze čisté virtuální metody v C++).

V případě jazyka C je použití třídy `Object` ve funkcích pracujících s uživatelsky definovanými typy částečně nahrazeno datovým typem `void *` (ukazatel na hodnotu libovolného datového typu). Při volání funkce vyžadující porovnávání je nutné v parametru namísto instance porovnávací třídy předat ukazatel na porovnávací funkci. Analogickým způsobem je potřeba pracovat s funkcemi, jejichž součástí je testování objektů na shodu (v duchu metody `equals()`) nebo výpis řetězcové reprezentace (po vzoru metody `toString()`) .V případě ostatních jazyků se od jakýchkoliv tříd, které budou využívat implementované knihovny, pro správnou funkci metod pracujících s objekty očekává dědění buď od třídy `Object`, nebo od jejího libovolného potomka.

3.3 Obalové třídy vestavěných datových typů

Pro každou z obalových tříd vestavěných datových typů jazyka Java (`Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean`, `Character`) byla vytvořena odpovídající třída (resp. struktura) v každém z ostatních jazyků, jejíž instance uchovávají hodnoty vestavěného datového typu daného jazyka, který je příslušnému typu jazyka Java nejbližší. V případě jazyků C, C++ a Pascal byly vždy upřednostněny číselné datové typy s garantovanými přesnými rozsahy hodnot.

Z důvodu zachování přehlednosti byly i v případě jazyků C a C++ rozlišeny datové typy pro znaky a pro 8-bitová celá čísla se znaménkem – pro znaky je používán vestavěný datový typ 8-bitového znaku, pro celá čísla pak číselný typ s garantovaným rozsahem. Pro logický typ byla v případě jazyka C využita definice typu `bool` poskytovaná standardní knihovnou. Kompletní

Tabulka 3.1: Typy jazyka Java a použité ekvivalenty v ostatních jazycích

Typ (Java)	**C/C++**	**C#**	**Pascal**
`byte`	`int8_t`	`Sbyte (sbyte)`	`ShortInt`
`short`	`int16_t`	`Int16 (short)`	`SmallInt`
`int`	`int32_t`	`Int32 (int)`	`LongInt`
`long`	`int64_t`	`Int64 (long)`	`Int64`
`float`	`float`	`Single (float)`	`Single`
`double`	`double`	`Double (double)`	`Double`
`boolean`	`bool`	`Boolean (bool)`	`Boolean`
`char`	`char`	`Char (char)`	`AnsiChar`

přehled typů jazyka Java a použitých odpovídajících typů v ostatních jazycích se nachází v tabulce 3.1 (poznámka: v případě jazyků C a C++ se názvy všech použitých typů shodují, proto nejsou uvedeny odděleně; v případě jazyka C# je vždy před závorkou uvedený název struktury reprezentující typ, v závorce pak následuje klíčové slovo používané jako alias této struktury).

Z metod poskytovaných obalovými třídami v jazyce Java byly pro všechny jazyky implementovány metody pro převod řetězcové reprezentace hodnoty na hodnotu příslušného datového typu (metody začínající řetězcem `parse`), metody provádějící opačný proces `toString()`, metody pro návrat hodnoty uložené v instanci objektu obalové třídy (končící řetězcem `Value()`) a několik dalších pomocných metod (zejména metody obalové třídy `Character` sloužící k rozpoznání číslic nebo písmen mezi předanými znaky `isDigit()`, resp. `isLetter()`). Implementované obalové třídy (moduly v případě jazyka C) dále definují konstanty představující hraniční hodnoty příslušných typů (například `MIN_VALUE` a `MAX_VALUE` pro nejmenší, resp. největší možnou číselnou hodnotu).

3.4 Rozhraní seznamů a třídy implementující seznamy

V knihovnách pro podporu datových struktur seznamů (`ArrayList` a `LinkedList`) byly u obou typů seznamů implementovány metody, resp. funkce `get()` pro operaci čtení prvků seznamu, `add()` pro jejich vkládání (včetně možnosti určení indexu), `set()` pro změnu hodnoty prvku, `remove()` pro jeho odstranění, `clear()` pro odstranění všech prvků najednou, `size()` pro zjištění počtu prvků, `isEmpty()` pro testování seznamu na zaplnění, `equals()` pro porovnání s jiným seznamem a `toString()` pro vytvoření tex-

tové reprezentace (podpisu) instance seznamu. Pokud byla v daném jazyku nutná (nebo výhodná) ruční implementace knihovny pro seznam reprezentující dynamické pole, byla zvolena výchozí počáteční (a zároveň minimální) velikost 10 prvků a koeficient změny velikosti 2 (tzn. vnitřní pole je v případě dosažení kritického počtu prvků buď zvětšeno na dvojnásobek, nebo zmenšeno na polovinu).

Jazyk C neposkytuje žádné vestavěné knihovny implementující seznamy, proto bylo nutno tyto datové struktury vytvořit svépomocí. Dynamické pole bylo implementováno s využitím funkce `realloc()` pro změnu velikosti alokované části paměti za běhu programu. Pro účely spojového seznamu byla vytvořena pomocná struktura `LinkedListNode`, která představuje uzel spojového seznamu uchovávající požadovanou hodnotu a odkazy na okolní uzly v seznamu (předchozí i následující, jedná se o obousměrně zřetězený seznam).

V jazyce C++ byla s malým množstvím přidaného kódu využita třída `std::vector` pro dynamické pole a třída `std::list` pro spojový seznam. Dodatečný kód vyžadovalo zejména iterování ve spojovém seznamu a v případě obou seznamů také implementace metody `toString()`.

Pro vytvoření obalových knihoven pro seznamy v jazyce C# byly využity vestavěné třídy jmenného prostoru `System.Collections.Generic`, jmenovitě třída `List` pro seznam implementovaný polem a třída `LinkedList` v kombinaci s třídou `LinkedListNode` pro spojový seznam. V případě prvního typu seznamu bylo pro vytvoření obalových metod zapotřebí minimum kódu (opět s výjimkou metody `toString()`), druhý typ vyžadoval větší množství kódu kvůli nutnosti práce s jednotlivými uzly ve struktuře spojového seznamu.

V jazyce Pascal byly pro zajištění větší podobosti chování s jazykem Java oba typy seznamů vytvořeny ručně po vzoru implementace v jazyce C. Odlišnosti obou implementací jsou minimální, pouze v případě spojového seznamu bylo namísto přímé práce s pamětí využito vestavěné struktury dynamického pole s možností měnit jeho velikost za běhu programu pomocí funkce `setLength()`.

V případě objektově orientovaných jazyků byla vytvořena i abstraktní třída `List` napodobující stejnojmenné rozhraní seznamů Javy, které umožňuje jednotný přístup k seznamům obou typů. V jazyce C je totéž řešeno (alespoň pro vytvořené knihovní funkce) předáváním ukazatelů na potřebné funkce knihoven seznamů v parametrech funkcí, které s těmito seznamy pracují.

3.5 Třídy pro manipulaci s poli a seznamy

Vytvořené knihovny `Arrays` pro složitější práci s poli podporují metody nebo funkce `binarySearch()` pro binární vyhledávání prvků zadané hodnoty, `fill()` pro naplnění všech prvků zadanou hodnotou, `sort()` pro řazení, `copyOf()` pro kopírování prvků z jednoho zadaného pole do jiného, `copyOfRange()` pro kopírování se zvolením rozsahu, `equals()` pro porovnávání a `toString()` pro výpis textové reprezentace z hodnot prvků. Knihovny `Collections` pro práci se seznamy pak podporují metody/funkce `binarySearch()`, `fill()` a `sort()` s obdobnou funkcionalitou, jako stejnojmenné funkce třídy `Arrays` v případě polí, a `copy()` plnící stejnou úlohu pro seznamy, jako `copyOf()` pro pole.

S výjimkou funkce pro nestabilní řazení `qsort()` a pro binární vyhledávání `bsearch()` vestavěné knihovny jazyka C uvedené operace nepodporují a byly implementovány vlastnoručně. Bez využití vestavěné funkce bylo implementováno i binární vyhledávání pro dosažení shodných výsledků pro hledání prvků nevyskytujících se v zadaném kontejneru. Vlastní implementaci vyžadovalo i stabilní řazení pro obecné struktury, pro které byl použit rekurzivní algoritmus Mergesort. Podobná situace nastala v případě jazyka C++ s tím rozdílem, že je k dispozici funkce pro stabilní řazení. K řazení tak byly použity funkce `std::sort()` a `std::stable_sort()`. V jazyce C# bylo kromě vestavěné metody `Sort()` třídy `System.Array`, která v duchu jazyka Java umožňuje stabilní i nestabilní řazení v závislosti na typu prvků předaného kontejneru (k řazení objektů je zvoleno stabilní řazení), dále možné použít i metodu `BinarySearch()` téže třídy pro binární vyhledávání se stejným chováním při neexistenci prvků, jako napodobovaná metoda v jazyce Java, tedy vracením záporných indexů těchto prvků, kdy absolutní hodnota každého indexu zmenšená o 1 určuje index v prohledávaném poli, na kterém by se prvek nacházel, pokud by existoval. Pro jazyk Pascal byly z důvodu slabé podpory operací s poli a seznamy prostřednictvím vestavěných knihoven vlastnoručně naprogramovány všechny operace obdobně jako v jazyce C, v tomto případě i včetně nestabilního řazení, kde byl implementován algoritmus Quicksort.

Uvedené obalové metody a funkce pro manipulaci s poli byly ve všech použitých jazycích navrženy v rámci možností obecně, tedy tak, aby pro kontejnery hodnot různých typů byl použit stejný kód. V jazycích C a Pascal byla typová genericita realizována vytvořením funkcí přebírajících jako parametry obecný ukazatel (ukazatel typu `void*`), který představuje pole libovolného typu, spolu s rozměry (počtem prvků a velikostí jednoho prvku v bajtech) a případně spolu s ukazatelem na potřebnou funkci pro zpracování prvků konkrétního typu (například pro porovnání dvou prvků nebo

sestavení textové reprezentace jednoho daného prvku), přičemž tyto funkce jsou v kódu knihovny předpřipraveny. Funkce/metody pro zpracování hodnot konkrétních datových typů pak obsahují pouze volání těchto obecných subrutin s předáním ukazatelů na odpovídající funkce pro zpracování jednotlivých prvků. V jazyce Pascal vyžadovaly navíc mírně odlišné zpracování metody pro manipulaci s poli objektů. Jazyky C++ a C# podporují šablony, resp. genericitu, která byla využita v obecných metodách namísto ukazatelů na data obecného typu, což přispělo k přehlednosti kódu.

3.6 Třídy pro operace s řetězci

Výčet naprogramovaných metod nebo funkcí, které slouží jako obdoby metod třídy `String` z Javy, je následující: `compareTo()`, `equals()`, `substring()`, `indexOf()`, `length()`, `trim()`, `toLowerCase()`, `toUpperCase()`, `charAt()`, `replace()`, `startsWith()`, `endsWith()`, `isEmpty()` a `toString()`. Konstruktory (resp. funkce pro vytváření struktur v jazyce C) podporují instancování třídy/struktury `String` buď z polí bajtů (kde číselné hodnoty jednotlivých bajtů reprezentují ASCII hodnoty znaků řetězce na příslušné pozici), nebo ze standardní reprezentace řetězce v daném jazyce (pole hodnot typu `char` v C, instance třídy `std::string` v C++ instance třídy `System.String` v C# a hodnota typu `AnsiString` v Pascalu).

Další částečně implementovanou knihovnou je `StringTokenizer`, sloužící k rozdělování řetězců na podřetězce podle určených oddělovačů, kde byly naprogramovány metody, resp. funkce `countTokens()`, `hasMoreTokens()` a `nextToken()`. Konstruktory umožňují rozdělení řetězce buď podle zadané množiny oddělovačů, nebo při jejím neuvedení podle množiny výchozí (tj. podle bílých znaků).

Poslední vytvořenou knihovnou pro manipulaci se seznamy je `StringBuilder`, která je určena k postupnému sestavování řetězce ze zadaných podřetězců, s metodami/funkcemi `append()` a `toString()`.

Pro všechny použité jazyky zde bylo ve velkém počtu případů možné využít stejně nebo podobně fungujících vestavěných metod nebo funkcí daného jazyka pro napodobení metod třídy `String`. Třídu `System.Text.StringBuilder` jazyka C# bylo možné použít beze změny, téměř beze změny pak i třídu `std::ostringstream` jazyka C++. Implementace od základu (jen s použitím funkcí pro alokaci a realokaci paměti) byla nutná v jazycích C a Pascal. Ne zcela triviální kód vyžadovala ve většině jazyků třída `StringTokenizer`. Výjimkou byl jazyk C#, kde byla implementace usnadněna díky dostupnosti metody `Split()` třídy `String` – s podřetězci pak bylo možno

zacházet jako s prvky pole. V jazycích C a C++ byla možná implementace s podporou funkcí jako je `strtok()`.

3.7 Třída pro matematické funkce

Z metod pro matematické funkce poskytovaných v jazyce Java jako členy třídy `Math` podporují obalové knihovny následující metody: `abs()`, `max()`, `min()`, `pow()`, `sqrt()`, `log()`, `log10()`, metody pro goniometrické funkce `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()` a metodu pro zaokrouhlování `round()`. V případě metod přetížených pro práci s hodnotami různých číselných datových typů byla implementována podpora všech. Kromě uvedených metod obsahují knihovny definice základních matematických konstant π a e. Vytvoření matematické knihovny bylo ve všech jazycích poměrně triviální vzhledem k podpoře uvedených matematických funkcí prostřednictvím vestavěných funkcí nebo metod vestavěných tříd.

3.8 Třídy pro uživatelský vstup a výstup

Naprogramované knihovny napodobující třídu `Scanner` jazyka Java pro načítání uživatelského vstupu obsahují metody s názvem začínajícím řetězcem „next“. Jejich funkcionalita byla ve všech jazycích implementována podobným způsobem – uživatelský vstup je vnitřně načítán po řádcích, přičemž aktuálně načtená řádka je uložena jako řetězec a při volání libovolné metody/funkce s výjimkou `nextLine()` je prohledána na výskyt nebílých znaků. První řetězec tvořený výhradně nebílými znaky je odstraněn z uloženého řetězce a v případě metody nebo funkce `next()` je vrácen jako načtený token. V ostatních metodách/funkcích s výjimkou `nextLine()` je před navrácením převeden na hodnotu odpovídajícího datového typu. Metoda/funkce `nextLine()` vrátí zbytek načtené řádky. Pokud načtená řádka neobsahuje další tokeny, při dalším zavolání jakékoliv metody/funkce „next“ je nejprve načtena nová řádka z uživatelského vstupu.

Pro implementaci obalových metod `print()` a `println()` pro výpis hodnot různých datových typů na obrazovku, které napodobují funkcionalitu stejnojmenných metod třídy `PrintStream` jazyka Java, bylo ve všech jazycích možné s minimem dodatečného kódu využít vestavěné knihovny – funkci `printf()` v jazyce C (s doplněním řetězce „\n“ v případě výpisů s ukončením řádky), funkci `std::cout` v jazyce C++ (spolu s funkcí `std::endl` pro výpisy s ukončením řádky), metody `Write()` a `WriteLine()` třídy `System.Console` v jazyce C# a funkce `Write()` a `WriteLn()` v jazyce

Pascal.

V případě výpisů číselných hodnot byl zvolen formát pro anglický jazyk (s tečkou „.“ jako desetinným oddělovačem). Formátování výstupu implementovaných subrutin se přesto může mírně lišit od jejich předloh v jazyce Java, to však pro účely této práce není považováno za zásadní nedostatek.

Po vzoru Javy byly ve všech jazycích s výjimkou C vytvořeny kromě tříd `Scanner` a `PrintStream` taktéž třídy `InputStream` a `System`. Třída `InputStream` nemá implementovánu žádnou funkcionalitu, je určena budoucím účelům a slouží jako typ objektu předávaného v konstruktoru třídy `Scanner` podle předlohy jazyka Java. Ve třídě `System` byly vytvořeny pouze dva statické atributy `in` a `out`, které uchovávají instance třídy `InputStream`, resp. `PrintStream` a umožňují tak vytváření instance třídy `Scanner` a volání metod pro výstup přesně v duchu jazyka Java. Náhradou v jazyce C jsou instance prázdných struktur `InputStream` a `PrintStream` uložené v globálních proměnných `System_in` a `System_out`.

4 Vytvoření aplikace pro správu zdrojového kódu

Tato kapitola se věnuje vytvoření desktopové aplikace sloužící ke správě zdrojových kódů implementovaných knihoven.

4.1 Analýza

Aplikace bude vytvořena pro běh na platformě Java ve verzi 1.8 (č. aktualizace 60 a vyšší).

4.1.1 Případy užití

Cílem vytvoření aplikace je umožnit načítání, zobrazování, úpravu a ukládání zdrojových kódů obalových knihoven podle zvoleného programovacího jazyka a zvolené Java třídy, jejíž rozhraní je předlohou pro vytvoření dané knihovny. Aplikace bude fungovat ve dvou režimech, mezi kterými bude možné se přepínat. Jedná se o režim zobrazení kódu a režim úpravy kódu.

V prvním jmenovaném režimu bude možné přidávat, upravovat a měnit položky v seznamech jazyků a tříd. Při označení položky v obou seznamech bude zobrazen zdrojový kód příslušné knihovny. Po přepnutí do druhého režimu bude zpřístupněna úprava aktuálně zobrazeného zdrojového kódu. To umožní aktualizaci uložených knihoven dle potřeby. Diagram případů užití aplikace se nachází v příloze A.

4.1.2 Struktura aplikace

Programový kód vytvořené aplikace bude strukturován podle MVC (Model-View-Controller) architektury. Model představuje aplikační vrstvu, která tvoří jádro aplikace určené ke čtení a manipulaci s daty. View a Controller tvoří grafické uživatelské rozhraní, které představuje prezentační vrstvu. View označuje kód aplikace popisující jednotlivé komponenty uživatelského rozhraní. Controller pak obsahuje metody pro obsluhu uživatelem vyvolaných událostí nad těmito komponentami a volání požadovaných operací jádra aplikace.

4.1.3 Vstup a výstup

Aplikace bude ukládat a načítat seznamy Java tříd, programovacích jazyků a zdrojové kódy knihoven do datových souborů ve formátu XML. Tento způsob byl zvolen jako vhodný kompromis mezi použitím databázového systému, které by bylo vzhledem k jednoduchosti hierarchie dat zbytečně komplikované a neumožňovalo by jednoduché přímé zobrazení a editaci dat, a ukládáním do prostých textových souborů, které by vyžadovalo vlastní návrh formátu dat od základu. Použití XML formátu vyžaduje pouze určení struktury XML souborů a umožňuje zobrazení a úpravu dat v čitelné formě v textovém editoru. Jak pro čtení, tak pro zápis dat v XML formátu bude použita technologie DOM (Document Object Model), která umožňuje v kódu programu přistupovat k obsahu XML dokumentu jako k elementům uspořádaným ve stromové struktuře.

4.1.4 Grafické uživatelské rozhraní

Součástí aplikace bude jednoduché grafické uživatelské rozhraní. Pro jeho vytvoření bude použita vestavěná grafická knihovna JavaFX. Možnou alternativou by mohla být starší grafická knihovna Swing, pro aplikaci MVC architektury se však JavaFX jeví jako vhodnější. Návrh okna GUI je na obrázku 4.1.

4.2 Popis implementace

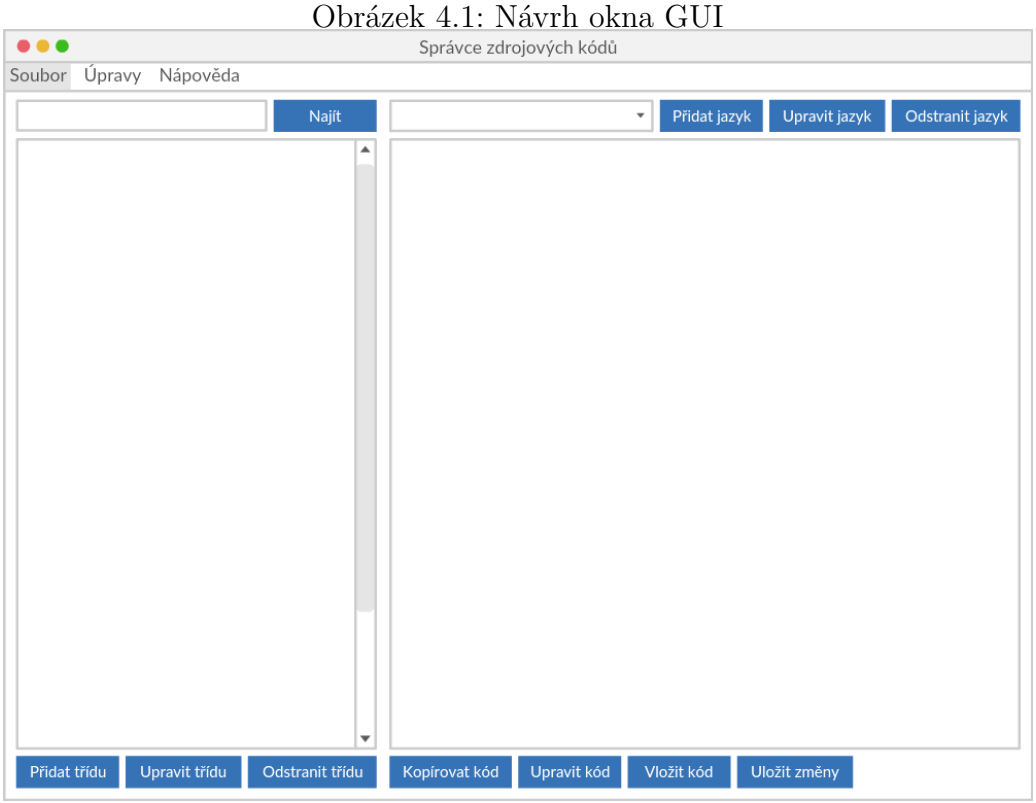
Aplikace byla naprogramována s použitím Oracle Java SE Development Kit 8 v integrovaném vývojovém prostředí Netbeans 8.0.2.

4.2.1 Struktura balíků a datového adresáře

Struktura balíků se zdrojovými kódy aplikace (které jsou součástí vytvořeného spustitelného `.jar` souboru) je následující:

- `<kořen>` – obsahuje `.java` soubory třídy `Main`, která je spouštěcí třídou programu, a třídy `Config`, která obsahuje veřejné konstanty použité na různých místech kódu a slouží tak jako konfigurační soubor aplikace
- `app` – obsahuje obecné rozhraní jádra pro napojení obsluhy GUI na jádro
- `app.xml` – obsahuje třídy jádra pro operace s XML soubory

Obrázek 4.1: Návrh okna GUI



- `gui` – obsahuje třídy pro obsluhu událostí GUI a pro vytváření dialogů
- `gui.fxml` – obsahuje `.fxml` dokument pro definici ovládacích prvků GUI
- `gui.strings` – obsahuje `.properties` soubor s texty popisů ovládacích prvků GUI
- `gui.strings.xml` – obsahuje soubor s texty GUI specifické pro operace s daty uloženými v XML souborech, jako jsou např. hlášení o chybách při čtení/zápisu souborů
- `gui.styles` – obsahuje .css soubor se styly aplikované na ovládací prvky GUI

Odděleně od zdrojových kódů se nachází adresář datových souborů. Jeho cesta je uvedena v konfiguračním souboru aplikace (výchozí název je `data`), není součástí spustitelného souboru aplikace, jeho obsah je aplikací načítán a upravován dle pokynů uživatele a má následující strukturu:

- `<kořen>` – obsahuje hlavní XML dokument se seznamy Java tříd a programovacích jazyků
- `classes` – obsahuje XML dokumenty se zdrojovými kódy obalových knihoven

Podrobný návrh aplikace z hlediska vzájemné závislosti tříd znázorňuje UML diagram tříd v příloze B.

4.2.2 Datová vrstva

Data aplikace sestávají z jednoho hlavního XML souboru `main.xml` a z adresáře `classes`. Soubor `main.xml` obsahuje dvojici seznamů – seznam podporovaných jazyků (pro účely práce tedy C, C++, C# a Pascal) představovaný elementem `languages` s položkami v podobě elementů `language`, a dále seznam Java tříd (element `classes` obsahující elementy `class` jako své položky), pro které jsou v uvedených jazycích vytvořeny rozhraním odpovídající knihovny.

Složka `classes` obsahuje XML soubory knihoven odpovídající seznamu knihoven v hlavním souboru. Název souboru knihoven odpovídá názvu příslušné třídy. Každý z těchto souborů pak obsahuje elementy `code` s vytvořeným zdrojovým kódem pro každý z jazyků uvedených v seznamu hlavního souboru. Programovací jazyk, ve kterém je napsán kód, který je obsahem daného elementu `code`, je určen atributem `lang`. Strukturu jednotlivých XML souborů graficky znázorňují diagramy v příloze C.

4.2.3 Aplikační vrstva

Aplikační vrstva tvoří jádro aplikace a představuje prostředníka mezi datovou a prezentační vrstvou. Umožňuje načítat a ukládat data v datové vrstvě a je oddělena od GUI, které využívá metody jejích tříd. Základem aplikační vrstvy je třída `app.xml.XmlManager` pro vytváření, čtení, úpravu a odstraňování datových XML souborů se zdrojovými kódy a odpovídajících záznamů v hlavním datovém XML souboru. Názvy XML elementů definuje výčtový typ `app.xml.XmlKeyword`.Hlavní třída této vrstvy `XmlManager` implementuje rozhraní `app.IDataManager`, pomocí něhož je ke třídě přistupováno v obsluze GUI. Tento návrh umožňuje případný snadný přechod na jinou datovou vrstvu (např. databázový systém).

Pro načítání a ukládání XML dokumentů s využitím technologie DOM je využito vestavěných tříd `DocumentBuilder` z balíku `javax.xml.parsers`, `Transformer` z balíku `javax.xml.transform` a `Document`, `NodeList`, `Node` a `Element` z balíku `org.w3c.dom`.

4.2.4 Prezentační vrstva

Hlavní okno GUI poskytuje uživateli snadný přístup k operacím aplikační vrstvy. Umožňuje prostřednictvím jádra spravovat seznamy tříd i programovacích jazyků, vyhledávat třídy v seznamu podle zadané počáteční části názvu a zobrazovat a upravovat uložené zdrojové kódy. Toto okno s odkazem na třídu obsluhy definuje FXML dokument `gui.fxml.Window.fxml`. Zde se nachází popis jednotlivých grafických komponent okna (hlavní menu, seznamy knihoven a jazyků, textové pole pro zobrazení a úpravu kódu a tlačítka pro spouštění operací s daty), jejich rozměry, textové popisky (resp. klíče k textům), počáteční stav, rozložení v okně a odkazy na metody obslužné třídy pro zpracování událostí nad těmito komponentami.

Řetězce popisů pro prvky GUI přiřazené jednotlivým klíčům jsou definovány v souborech podbalíků `gui.strings` a `gui.strings.xml`, koncovka názvů „_cs“ značí texty pro češtinu. Oddělení textů umožňuje snadný překlad GUI aplikace do dalších jazyků a jejich případné přepínání za běhu programu. Vzhled prvků GUI je dále upraven v souboru stylů v podbalíku `gui.styles`. To umožňuje změnu vzhledu z jednoho místa kódu (a případně pro více oken aplikace). Soubor stylů je využit zejména pro nastavení písma kódu v textovém poli (výchozí písmo má různou šířku znaků, ale pro zdrojový kód je vhodné použití strojového písma s jednotnou šířkou).

Obsluhu událostí okna obstarává třída `gui.WindowController`. Součástí metod obsluhy je obvykle validace vstupních dat zadaných prostřednictvím dialogů (zobrazování různých typů dialogů pro zadávání uživatelského

vstupu nebo pro výpisy hlášení zprostředkovává třída `gui.DialogFactory`) a následné volání metod instance třídy jádra, která implementuje rozhraní `IDataManager` – v případě ukládání dat do XML souborů se tedy jedná o vytvořenou třídu `XmlManager`. Vlastní provádění obsluhy je řešeno s využitím instancí vestavěné třídy `javafx.concurrent.Task`, která slouží k definici operací spouštěných na pozadí (tj. v novém vlákně) a k následnému zobrazení výsledku do grafických prvků okna v hlavním vlákně GUI. Díky tomu je možné provádět potenciálně časově náročné operace, aniž by okno GUI přestávalo odpovídat do okamžiku jejich dokončení, a následně správně zobrazovat výsledky těchto operací v GUI.

Inicializace objektu pro správu dat a objektu pro vytváření dialogů, jejich propojení s kontrolerem hlavního okna `WindowController` a zobrazení tohoto okna probíhá bezprostředně po spuštění aplikace v hlavní třídě `Main`, přičemž jsou použity cesty ke konkrétním částem aplikace (k FXML dokumentu okna a ke třídě správce dat) a další hodnoty výchozího nastavení aplikace (např. jazyk uživatelského rozhraní) uložené jako statické konstanty v konfigurační třídě `Config`. Návrh poskytuje do budoucna možnost jednoduché konfigurace a výměny jednotlivých vrstev aplikace bez nutnosti zásahu (nebo s nutností pouze minimálních zásahů) do ostatních vrstev.

5 Testování aplikace a knihoven

Tato kapitola shrnuje způsoby testování jednotlivých částí vytvořeného programového vybavení.

5.1 Testování knihoven

Funkčnost vytvořených knihoven byla ověřena pomocí jednotkových testů. Třídy/moduly pro jednotkové testování knihoven jsou přiloženy na CD.

5.1.1 Podmínky a způsob testování

Spouštění testů probíhalo na následujících 64-bitových operačních systémech:

1. Windows 10 Home (na tomto systému probíhal i vývoj)
2. Ubuntu Linux 14.04 (s výjimkou testování jazyka C#)

Základní princip otestování knihoven byl pro všechny jazyky shodný – ověřování funkčnosti probíhalo pomocí automaticky spouštěných jednotkových testů metod nebo funkcí, které tvoří rozhraní jednotlivých knihoven. Tyto testy byly sdruženy do testovacích modulů, resp. tříd, které byly vytvořeny zvlášť pro každou knihovnu. Testování v jednotlivých jazycích však probíhalo mírně odlišným způsobem v závislosti na možnostech poskytovaných daným jazykem a použitým integrovaným vývojovým prostředím (dále jen IDE). Specifický případ pak představovalo testování knihoven pro vstupní a výstupní (dále jen IO) operace.

5.1.2 Testy v jazyce C

Pro testování vytvořených knihoven v jazyce C byl vytvořen vlastní pomocný modul `test.c` s funkcemi deklarovanými v hlavičkovém souboru `test.h`. Tento modul obsahuje funkce sloužící k inicializaci testování pro danou knihovnu, k zaregistrování jednotlivých funkcí testovacího modulu (což je implementováno jako uložení ukazatelů na tyto funkce do kontejneru), k postupnému spuštění registrovaných funkcí s výpisy výsledků na standardní

výstup, a také modul obsahuje funkce k porovnávání předaných výsledných hodnot různých datových typů, které představují výstupy operací v testovaných knihovnách, s předanými očekávanými výsledky těchto operací (názvy těchto porovnávacích funkcí začínají řetezcem „assert“). S využitím těchto funkcí bylo dosaženo jednoduchého napodobení způsobu testování pomocí běžných knihoven třetích stran pro jednotkové testování, jako je například knihovna JUnit pro jazyk Java. Použitým IDE k implementaci i testování knihoven pro jazyk C byla aplikace Netbeans 8.0.2 s překladačem GCC 4.8.3 (určeném pro prostředí Cygwin v případě testování na OS Windows).

5.1.3 Testy v jazyce C++

Při testování C++ knihoven byl dodržen stejný postup jako v případě jazyka C, při vytváření pomocného modulu byly pouze využity dodatečné možnosti jazyka C++, např. funkce pro testování byly sdruženy jako metody třídy `Test` a ty z metod, které slouží pro porovnávání výsledné hodnoty s očekávanou, využívají možnost přetížení.

5.1.4 Testy v jazyce C#

Jak implementace, tak i testování knihoven pro jazyk C# probíhaly na platformě Microsoft .NET Framework 4.6 s pomocí IDE Microsoft Visual Studio Community 2015. Zde bylo možné využít standardní knihovny pro jednotkové testy s integrovanou podporou v použitém IDE (celým názvem Microsoft.VisualStudio.QualityTools.UnitTestFramework), což znatelně usnadnilo vytvoření testovacích tříd bez nutnosti implementace vlastní podpůrné testovací knihovny.

5.1.5 Testy v jazyce Pascal

Implementace i testování knihoven pro jazyk Pascal byly prováděny v IDE Lazarus verze 1.4.4 s použitím překladače FreePascal 2.6.4. Stejně jako v případě jazyka C#, i zde bylo možné využít existující podpory jednotkového testování prostřednictvím vestavěné knihovny FPCUnit s integrovanou podporou ze strany IDE a nebylo tak potřeba vytvářet vlastní pomocné knihovny.

5.1.6 Testy IO operací v jednotlivých jazycích

Nad knihovnami pro načítání uživatelského vstupu a výpis výstupu (tedy knihovny `Scanner` a `PrintStream`) nebylo možné provádět přímé automatické jednotkové testy stejným způsobem, jako v případě ostatních knihoven.

Jednou z potenciálních možností je přesměrování vstupu, resp. výstupu ze standardního (klávesnice/obrazovka) do souborového, což umožňuje v případě vstupu předpřipravení testovacích dat a v případě výstupu kontrolu vypsaných dat. Operace se soubory však rozhraní vytvořených knihoven neumožňuje, protože se předpokládá pouze standardní vstup a výstup. Další případnou možností je ruční testování (tedy vytvoření triviálních programů používajících rozhraní knihoven pro IO operace a jejich spouštění s ručním zadáváním testovacích dat z klávesnice, resp. s ruční kontrolou vypsaných dat na obrazovku), takovéto řešení by však bylo značně neefektivní vzhledem k množství testovaných funkcí a metod.

Jako řešení tohoto specifického případu testování byla zvolena částečná kombinace obou uvedených principů s podporou vytvořeného skriptu pro automatizované spouštění testů. V každém použitém jazyce byla vytvořena dvojice jednoduchých utilit, z nichž jedna slouží ke spouštění podprogramů testovací třídy/modulu pro knihovnu Scanner a v případě druhé jsou spouštěny podprogramy testovací třídy/modulu určené k testování knihovny PrintStream. Testovací metoda/funkce k zavolání je zvolena podle pořadí jejího výskytu v programovém kódu testovací sady, které je předáno volajícímu programu při jeho spuštění jako parametr příkazové řádky.

Automatizaci postupného spouštění testů zajišťuje jednoduchý skript naprogramovaný v jazyce PHP (použitá verze interpretu je 5.6.). Tento skript spouští testovací utilitu na zadané cestě vestavěnou funkcí `exec()`. V případě testování vstupu je utilita spuštěna s přesměrováním standardního vstupu zvenčí na příslušný předpřipravený soubor s testovacími daty a následným zobrazením výstupu utility. Při testování výstupu jsou řetězce vypisované utilitou ukládány skriptem do proměnné, jejíž obsah je poté porovnáván s očekávaným řetězcem výstupu.

5.1.7 Výsledky testování knihoven

Testování odhalilo řadu převážně drobných chyb v implementaci knihoven, z nichž valná většina byla postupně odstraněna. Výjimkou jsou drobné odlišnosti ve výstupech matematických funkcí pro desetinná čísla, v převodu desetinných čísel do řetězců a zpět a ve formátování výpisů desetinných čísel na obrazovku, které se odstranit nepodařilo, nicméně výsledky metod/funkcí pro matematické výpočty se s odpovídajícími metodami třídy `Math` jazyka Java shodují s tolerancí maximálního rozdílu mezi očekávanou a skutečnou hodnotou 10^{-10} (v případě jazyka C# až 10^{-20}). Dále se nepodařilo odstranit problém, který způsoboval pád programu při pokusech o konverzi obecných objektů třídy `Object` na porovnávatelné objekty třídy `Comparable`

s využitím genericity v jazyce C# (v jazyce C++ se v obdobných situacích objevily problémy při překladu). Z tohoto důvodu byla podpora genericity ze tříd napodobujících rozhraní jazyka Java pro porovnávání objektů odstraněna a metody těchto tříd tak vyžadují přetypování porovnávaných objektů. Zbylou funkcionalitu knihoven se díky automatizovaným testům podařilo odladit tak, aby výstupy jejich metod/funkcí odpovídaly výstupům příslušných metod knihoven jazyka Java.

5.2 Testování aplikace

Pro jednotlivé části aplikace byly použity různé způsoby testování.

5.2.1 Testy jádra

Jádro aplikace bylo otestováno s využitím knihovny pro jednotkové testy JUnit 4.10 a IDE Netbeans 8.0.2, které pro použitou testovací knihovnu obsahuje integrovanou podporu. Za účelem otestování byly vytvořeny dvě testovací třídy – třída `XmlManagerTestPositive` pro pozitivní testy (testy chování třídy jádra v reakci na volání metod jejího rozhraní s předáním platných vstupů) a třída `XmlManagerTestNegative` pro negativní (testy chování při neplatných vstupech). Před každým jednotlivým testem (resp. před spuštěním každé testovací metody) jsou vstupní data obnovena do původního (výchozího) stavu. Uvedené třídy pro jednotkové testování jádra aplikace jsou přiloženy na CD.

Pozitivní testy zahrnují test inicializace objektu správy dat, testy načtení seznamu Java tříd a seznamu programovacích jazyků, testy přidávání, upravování a odstraňování tříd nebo jazyků a v neposlední řadě testy ukládání a načítání zdrojových kódů. Dále třída pozitivních testů obsahuje metodu pro komplexní otestování všech operací prováděných střídavě a opakovaně.

Negativní testy sestávají z testů inicializace při neexistujícím hlavním souboru nebo složce souborů knihoven, testů načtení duplicitních nebo chybějících seznamů tříd nebo jazyků, testů přidávání, upravování a odstraňování tříd nebo jazyků za různých nestandardních situací (duplicitní nebo chybějící seznam v hlavním souboru, existující soubor knihovny se zadaným názvem při operaci přidání nebo úpravy nebo naopak chybějící při operaci úpravy nebo odstranění) a testů uložení nebo načtení kódu při chybějícím XML elementu s kódem pro danou třídu v daném jazyce. Očekávanou reakcí jádra na neplatná vstupní data je vyhození výjimky, která obsahuje odpovídající chybové hlášení (které je při běhu aplikace zobrazeno v GUI

prostřednictvím dialogu). Při testech jsou porovnávána skutečná chybová hlášení s požadovanými.

5.2.2 Testy uživatelského rozhraní

Uživatelské rozhraní aplikace bylo ověřeno ručním testováním s porovnáváním skutečné funkčnosti ovládacích prvků hlavního okna proti následujícím požadavkům:

1. Tlačítka a položky menu `Přidat třídu`, `Upravit třídu` a `Odstranit třídu` jsou aktivní právě tehdy, je-li vybrána položka v seznamu Java tříd (je možné označit k výběru maximálně jednu položku seznamu).

2. Tlačítka a položky menu `Přidat jazyk`, `Upravit jazyk` a `Odstranit jazyk` jsou aktivní právě tehdy, je-li vybrána položka v nabídce programovacích jazyků (je možné vybrat maximálně jednu položku nabídky).

3. Tlačítka a položky menu `Kopírovat kód`, `Upravit kód`, `Vložit kód` a `Uložit změny` jsou aktivní právě tehdy, je-li vybrána současně jak položka v seznamu Java tříd, tak položka v nabídce programovacích jazyků.

4. Po kliknutí na tlačítko nebo položku menu `Přidat třídu` se zobrazí dialog pro zadání názvu nové třídy. Po zadání a potvrzení platného názvu se objeví položka s názvem nové třídy v seznamu tříd a tato položka je označena.

5. Po kliknutí na tlačítko nebo položku menu `Upravit třídu` se zobrazí dialog pro zadání nového názvu existující třídy. Po zadání a potvrzení platného názvu se změní popis položky s názvem vybrané třídy v seznamu tříd a tato položka zůstává označena.

6. Po kliknutí na tlačítko nebo položku menu `Odstranit třídu` se zobrazí dialog pro potvrzení odstranění existující třídy. Po potvrzení zmizí položka s názvem vybrané třídy ze seznamu tříd a označení položek v seznamu je vynulováno.

7. Po kliknutí na tlačítko nebo položku menu `Přidat jazyk` se zobrazí dialog pro zadání názvu nového jazyka. Po zadání a potvrzení platného názvu se objeví položka s názvem nového jazyka v nabídce jazyků a tato položka je označena.

8. Po kliknutí na tlačítko nebo položku menu `Upravit jazyk` se zobrazí dialog pro zadání nového názvu existujícího jazyka. Po zadání a potvrzení platného názvu se změní popis položky s názvem vybraného jazyka v nabídce jazyků a tato položka zůstává označena.

9. Po kliknutí na tlačítko nebo položku menu `Odstranit jazyk` se zobrazí dialog pro potvrzení odstranění existujícího jazyka. Po potvrzení zmizí položka s názvem vybraného jazyka z nabídky jazyků a označení položek v nabídce je vynulováno.

10. Po kliknutí na položku menu `Ukončit program` dojde k ukončení aplikace.

11. Po kliknutí na tlačítko nebo položku menu `Kopírovat kód` se zkopíruje do schránky zdrojový kód zobrazený v textovém poli.

12. Po kliknutí na tlačítko nebo položku menu `Upravit kód` se umožní editace kódu zobrazeného v textovém poli, popis tohoto tlačítka/položky se změní na `Zrušit úpravy`, aktivují se tlačítka pro vložení kódu a uložení změn a deaktivují se seznam tříd s nabídkou jazyků a tlačítky pro operace s jejich položkami.

13. Po kliknutí na tlačítko nebo položku menu `Zrušit úpravy` při úpravě kódu se zobrazí dialog pro potvrzení zrušení změn v kódu. Po potvrzení dialogu se v textovém okně zobrazí původní verze kódu načtená z datových souborů a nastavení grafických komponent okna se vrátí do stavu před zahájením úprav kliknutím na tlačítko `Upravit kód`.

14. Potvrzovací dialog zrušení změn při úpravě kódu se zobrazí i v případě pokusu o ukončení aplikace.

15. Po kliknutí na tlačítko nebo položku menu `Vložit kód` při úpravě kódu se nahradí zdrojový kód upravovaný v textovém poli textem vloženým ze schránky.

16. Po kliknutí na tlačítko nebo položku menu `Uložit změny` při úpravě kódu se uloží nová verze kódu v textovém poli do datových souborů a nastavení grafických komponent okna se vrátí do stavu před zahájením úprav kliknutím na tlačítko `Upravit kód`.

17. Po kliknutí na položku menu `O programu` se zobrazí dialogové okno obsahující stručný popis aplikace.

18. Po kliknutí na tlačítko `Najít` se změní výběr položky v seznamu tříd na první nalezenou položku, jejíž název začíná na řetězec zadaný do vyhledávacího políčka bez ohledu na velikost písmen.

5.2.3 Výsledky testování aplikace

Při testování byly postupně zjištěny nedostatky jak ve funkcionalitě jádra, tak uživatelského rozhraní aplikace. Všechny odhalené chyby se nakonec podařilo úspěšně odstranit. Jádro aplikace po odladění požadovaným způsobem zpracovává korektní vstupní data pro všechny prováděné operace a na nekorektní vstupy reaguje výjimkami s chybovými hláškami podle očekávání. Uživatelské rozhraní taktéž reaguje na akce uživatele předpokládaným způsobem podle uvedeného seznamu požadavků.

6 Závěr

Dosažené výsledky z velké části odpovídají původním představám. Po prostudování jednotlivých programovacích jazyků a jejich vestavěných knihoven byly naprogramovány knihovny s rozhraním ve stylu vestavěných tříd jazyka Java, které poskytuje funkcionalitu podle požadavků vedoucího práce, pro správu zdrojových kódů těchto knihoven byla vytvořena desktopová aplikace a obě části programového vybavení (tedy jak knihovny, tak aplikace) byly důkladně otestovány a odladěny, aby byla zajištěna jejich správná funkčnost.

Lze tedy konstatovat, že se podařilo splnit všechny body zadání práce, ačkoliv některé části programového vybavení nebyly dokončeny v rozsahu podle původních představ. To se týká například aplikace pro správu knihoven, kde by bylo vhodné doimplementovat zvýraznění syntaxe zdrojového kódu v textovém poli nebo knihoven pro výstup na obrazovku, kde byla původně plánována implementace formátování výstupu podle zadaných formátovacích řetězců (po vzoru metody `printf()` třídy `java.io.PrintStream`).

Práce ukázala jednu z možností, jak je možné usnadnit převod programového kódu mezi různými programovacími jazyky. Cílem dalších prací podobného charakteru by mohla být automatizace tohoto procesu včetně převodu klíčových slov a syntaktických konstrukcí daného programovacího jazyka na jejich ekvivalenty v cílovém jazyce.

Literatura

[1] SHARON ZAKHOUR, J. R. I. R. T. R. M. H. S. H. *Java 6 Výukový kurz.* Computer Press, 2007. ISBN 978-80-251-1575-6.

[2] *Java SE 7 Virtual Machine (VM)-related APIs & Developer Guides* [online]. [cit. 1.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/technotes/guides/vm/index.html`.

[3] *Java SE 7 Java Compiler (javac)-related APIs and Developer Guides* [online]. [cit. 1.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/technotes/guides/javac/index.html`.

[4] *java* [online]. [cit. 1.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/technotes/tools/windows/java.html`.

[5] *Object (Java Platform SE 7)* [online]. [cit. 1.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html`.

[6] HEROUT, P. *Učebnice jazyka C.* KOPP, 1994. ISBN 80-85828-21-9.

[7] *The GNU C Reference Manual* [online]. [cit. 1.12.2015]. Dostupné z: `http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html`.

[8] PRATA, S. *Mistrovsví v C++.* Computer Press, 2001. ISBN 80-7226–339-0.

[9] *Classes (I) - C++ Tutorials* [online]. [cit. 1.12.2015]. Dostupné z: `http://www.cplusplus.com/doc/tutorial/classes/`.

[10] SHARP, J. *Microsoft Visual C# 2010 Krok za krokem.* Computer Press, 2010. ISBN 978-80-251-3147-3.

[11] *Visual C# Language (C#)* [online]. [cit. 2.12.2015]. Dostupné z: `https://msdn.microsoft.com/en-us/library/aa287558(v=vs.71).aspx`.

[12] *Common Language Runtime (CLR)* [online]. [cit. 2.12.2015]. Dostupné z: `https://msdn.microsoft.com/cs-cz/library/8bs2ecf4(v=vs.110).aspx`.

[13] *Compiling to MSIL* [online]. [cit. 2.12.2015]. Dostupné z: `https://msdn.microsoft.com/en-us/library/c5tkafs1(v=vs.100).aspx`.

[14] *Compiling MSIL to Native Code* [online]. [cit. 2.12.2015]. Dostupné z: `https://msdn.microsoft.com/en-us/library/ht8ecch6(v=vs.100).aspx`.

[15] *Built-in Data Types* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/cs7y5x0x(v=vs.90).aspx.

[16] SATRAPA, P. *Pascal pro zelenáče*. Neocortex, 2000. ISBN 80-86330-03-6.

[17] *About the Pascal language* [online]. [cit. 2.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refli6.html#x7-6000.

[18] *Primitive Data Types (The Java^{TM} Tutorials > Learning the Java Language > Language Basics)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html.

[19] *Byte (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Byte.html.

[20] *Short (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Short.html.

[21] *Integer (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html.

[22] *Long (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Long.html.

[23] *Float (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Float.html.

[24] *Double (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html.

[25] *Boolean (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Boolean.html.

[26] *Character (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Character.html.

[27] *<cstdint> (stdint.h) - C++ Reference* [online]. [cit. 21.03.2016]. Dostupné z: http://www.cplusplus.com/reference/cstdint/.

[28] *Variables and types - C++ Tutorials* [online]. [cit. 3.12.2015]. Dostupné z: http://www.cplusplus.com/doc/tutorial/variables/.

[29] *sbyte (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/d86he86x.aspx.

[30] *short (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/ybs77ex4.aspx.

[31] *int (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/5kzh1b5w.aspx.

[32] *long (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/ctetwysk.aspx.

[33] *float (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/b1e65aza.aspx.

[34] *double (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/678hzkk9.aspx.

[35] *bool (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/c8f5xwh7.aspx.

[36] *char (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/x9h8tsay.aspx.

[37] *byte (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/5bdb6693.aspx.

[38] *ushort (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/cbf1574z.aspx.

[39] *uint (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/x0sksh43.aspx.

[40] *ulong (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/t98873t4.aspx.

[41] *decimal (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/364x0z75.aspx.

[42] *string (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/362314fe.aspx.

[43] *object (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/9kkx3h3c.aspx.

[44] *3 Types* [online]. [cit. 3.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refch3.html#x25-240003.

[45] *Base types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse12.html#x26-250003.1.

[46] *Character types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse13.html#x29-320003.2.

[47] *Ordinal types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu5.html#x27-260003.1.1.

[48] *Ordinal types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu5.html#x27-280003.1.1.

[49] *Real types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu6.html#x28-310003.1.2.

[50] *Char or AnsiChar* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu7.html#x30-330003.2.1.

[51] *WideChar* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu8.html#x31-340003.2.2.

[52] *Arrays (The Java^{TM} Tutorials > Learning the Java Language > Language Basics)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html.

[53] *Arrays (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html.

[54] *Collection (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html.

[55] *ArrayList (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html.

[56] *LinkedList (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html.

[57] *HashSet (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html.

[58] *HashMap (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html.

[59] *List (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/List.html.

[60] *Collections (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html.

[61] *Arrays - C++ Tutorials* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/doc/tutorial/arrays/.

[62] *Pointers - C++ Tutorials* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/doc/tutorial/pointers/.

[63] *malloc - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/malloc/.

[64] *calloc - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/calloc/.

[65] *vector - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/vector/vector/.

[66] *list - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/list/list/.

[67] *set - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/set/set/.

[68] *map - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/map/map/.

[69] *Arrays Tutorial (C#)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/aa288453(v=vs.71).aspx.

[70] *Array – třída (System)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.array(v=vs.110).aspx.

[71] *List(T) – třída (System.Collections.Generic)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/6sh2ey19(v=vs.110).aspx.

[72] *LinkedList(T) – třída (System.Collections.Generic)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/he2s3bh7(v=vs.110).aspx.

[73] *Structured Types* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse14.html#x38-480003.3.

[74] *Arrays* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu15.html#x39-510003.3.1.

[75] *Arrays* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu15.html#x39-520003.3.1.

[76] *Pointers* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse15.html#x43-610003.4.

[77] *TList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tlist.html.

[78] *TFPList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/classes/tfplist.html.

[79] *TFPGList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/fgl/tfpglist.html.

[80] *TFPGObjectList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/fgl/tfpgobjectlist.html.

[81] *Strings (The Java^{TM} Tutorials > Learning the Java Language > Numbers and Strings)* [online]. [cit. 6.12.2015]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/data/strings.html.

[82] *String (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html.

[83] *StringBuffer (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html.

[84] *StringBuilder (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html.

[85] *StringTokenizer (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html.

[86] *Character sequences - C++ Tutorials* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/doc/tutorial/ntcs/.

[87] *<cstring> (string.h) - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstring/.

[88] *<cstdlib> (stdlib.h) - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/.

[89] *<string> - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/reference/string/.

[90] *string - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/reference/string/string/?kw=string.

[91] *String Class (System)* [online]. [cit. 7.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx.

[92] *String.Split Method (String[], StringSplitOptions) (System)* [online]. [cit. 7.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/tabh47cf(v=vs.110).aspx.

[93] *Single-byte String types* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu10.html#x33-360003.2.4.

[94] *Single-byte String types* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu10.html#x33-370003.2.4.

[95] *Single-byte String types* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu10.html#x33-380003.2.4.

[96] *Reference for unit 'System': Procedures and functions* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/system/index-5.html.

[97] *TStringList* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tstringlist.html.

[98] *TStrings* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tstrings.html.

[99] *Comparable (Java Platform SE 7)* [online]. [cit. 8.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html.

[100] *Comparator (Java Platform SE 7)* [online]. [cit. 8.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html.

[101] *qsort - C++ Reference* [online]. [cit. 8.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/qsort/.

[102] *bsearch - C++ Reference* [online]. [cit. 8.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/bsearch/.

[103] *sort - C++ Reference* [online]. [cit. 9.12.2015]. Dostupné z: http://www.cplusplus.com/reference/algorithm/sort/.

[104] *stable_sort - C++ Reference* [online]. [cit. 9.12.2015]. Dostupné z: http://www.cplusplus.com/reference/algorithm/stable_sort/.

[105] *binary_search - C++ Reference* [online]. [cit. 9.12.2015]. Dostupné z: http://www.cplusplus.com/reference/algorithm/binary_search/.

[106] *IComparable(T) – rozhraní (System)* [online]. [cit. 9.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/4d7sx9hd(v=vs.110).aspx.

[107] *IComparer(T) – rozhraní (System.Collections.Generic)* [online]. [cit. 9.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/8ehhxeaf(v=vs.110).aspx.

[108] *Comparison(T) – delegát (System)* [online]. [cit. 9.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/tfakywbh(v=vs.110).aspx.

[109] *TList.Sort* [online]. [cit. 9.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tlist.sort.html.

[110] *TFPList.Sort* [online]. [cit. 9.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/classes/tfplist.sort.html.

[111] *TListSortCompare* [online]. [cit. 9.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/classes/tlistsortcompare.html.

[112] *Math (Java Platform SE 7)* [online]. [cit. 10.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html.

[113] *<cmath> (math.h) - C++ Reference* [online]. [cit. 10.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cmath/.

[114] *abs - C++ Reference* [online]. [cit. 10.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cmath/abs/.

[115] *Math – třída (System)* [online]. [cit. 10.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.math(v=vs.110).aspx.

[116] *Reference for unit 'math': Procedures and functions* [online]. [cit. 10.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/math/index-5.html.

[117] *Scanner (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html.

[118] *InputStream (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html.

[119] *System (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/System.html.

[120] *OutputStream (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html.

[121] *PrintStream (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html.

[122] *<cstdio> (stdio.h) - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdio/.

[123] *scanf - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdio/scanf/.

[124] *printf - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdio/printf/.

[125] *<iostream> - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/iostream/.

[126] *cin - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/iostream/cin/.

[127] *cout - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/iostream/cout/.

[128] *Console – třída (System)* [online]. [cit. 12.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.console(v=vs.110).aspx.

[129] *Convert – třída (System)* [online]. [cit. 12.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.convert(v=vs.110).aspx.

[130] *Read* [online]. [cit. 12.12.2015]. Dostupné z:
`http://www.freepascal.org/docs-html/rtl/system/read.html`.

[131] *Write* [online]. [cit. 12.12.2015]. Dostupné z:
`http://www.freepascal.org/docs-html/rtl/system/write.html`.

[132] *ReadLn* [online]. [cit. 12.12.2015]. Dostupné z:
`http://www.freepascal.org/docs-html/rtl/system/readln.html`.

[133] *WriteLn* [online]. [cit. 12.12.2015]. Dostupné z:
`http://www.freepascal.org/docs-html/rtl/system/writeln.html`.

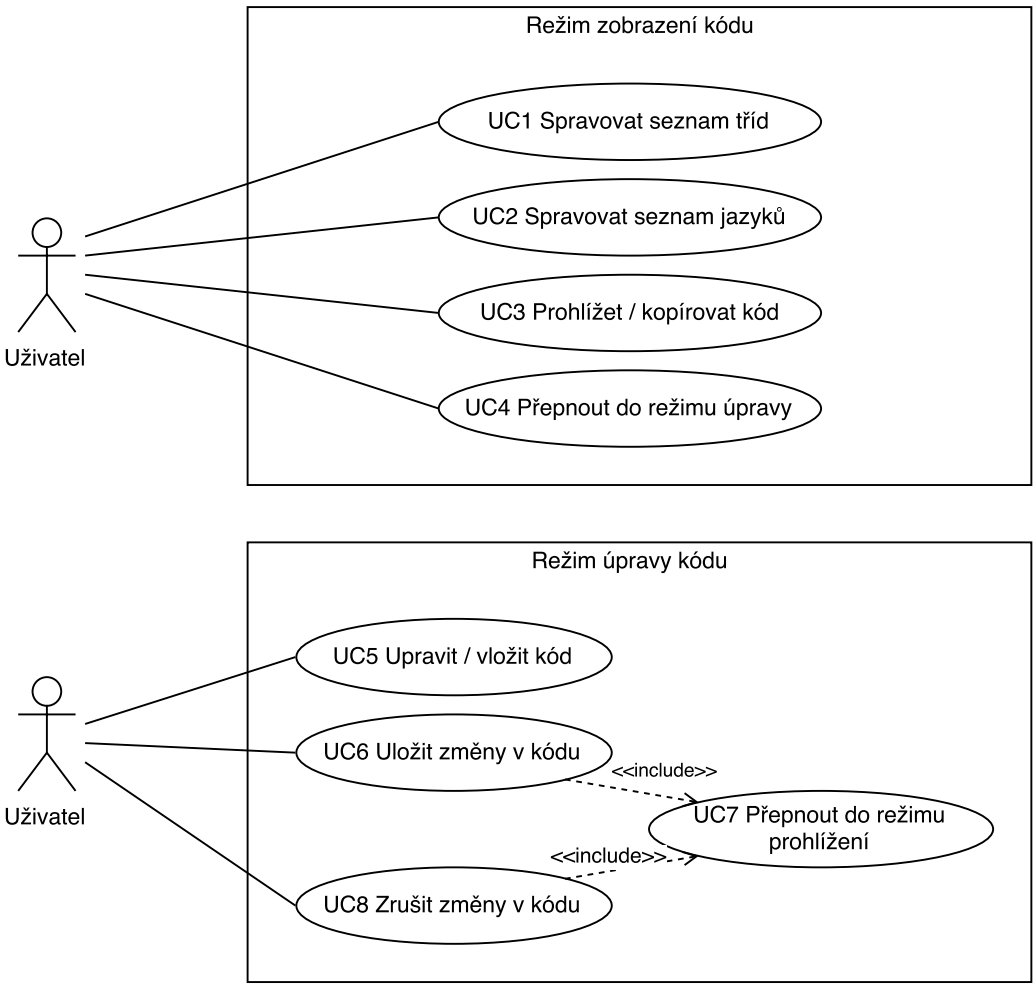
Použité zkratky a značení

1. IDE – integrované vývojové prostředí
2. IO – vstupní a výstupní
3. OO – objektově orientovaný
4. knihovna – část programového kódu logicky sdružující nějakou funkcionalitu, v kontextu vytvořeného kódu je tento pojem obvykle totožný s pojmem třída (pro jazyky umožňující OO přístup) nebo zahrnuje modul výkonného kódu spolu s příslušným hlavičkovým souborem (pro jazyk C)
5. subrutina, podprogram – jednotné označení pro metodu třídy (v případě OO přístupu) nebo samostatnou funkci (v případě přístupu procedurálního, tedy především v jazyce C)
6. kontejner – společné pojmenování pro libovolnou datovou strukturu k uchovávání většího počtu hodnot stejného typu (statická pole, seznamy implementované jako pole s dynamicky se měnící velikostí a spojové seznamy)
7. řetězec – uspořádaná posloupnost znaků představující text (které mohou být jedno- i vícebajtové v závislosti na použitém jazyce)

PŘÍLOHY

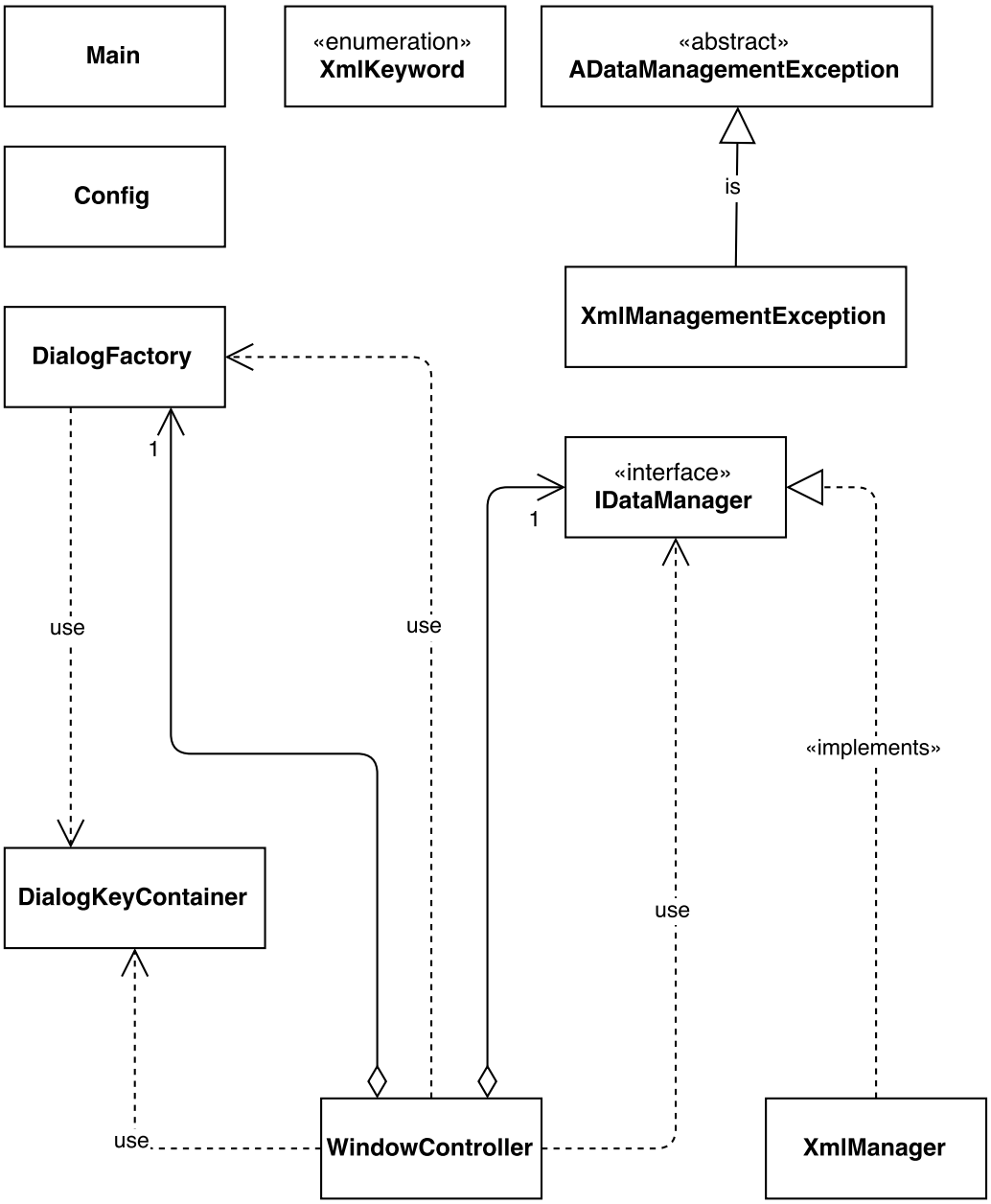
A Diagram případů užití aplikace

Tato příloha obsahuje diagram případů užití aplikace pro správu zdrojového kódu.


Režim zobrazení kódu
UC1 Spravovat seznam tříd
UC2 Spravovat seznam jazyků
UC3 Prohlížet / kopírovat kód
UC4 Přepnout do režimu úpravy
Uživatel
Režim úpravy kódu
UC5 Upravit / vložit kód
UC6 Uložit změny v kódu
<<include>>
UC7 Přepnout do režimu prohlížení
<<include>>
UC8 Zrušit změny v kódu
Uživatel

B UML diagram tříd aplikace

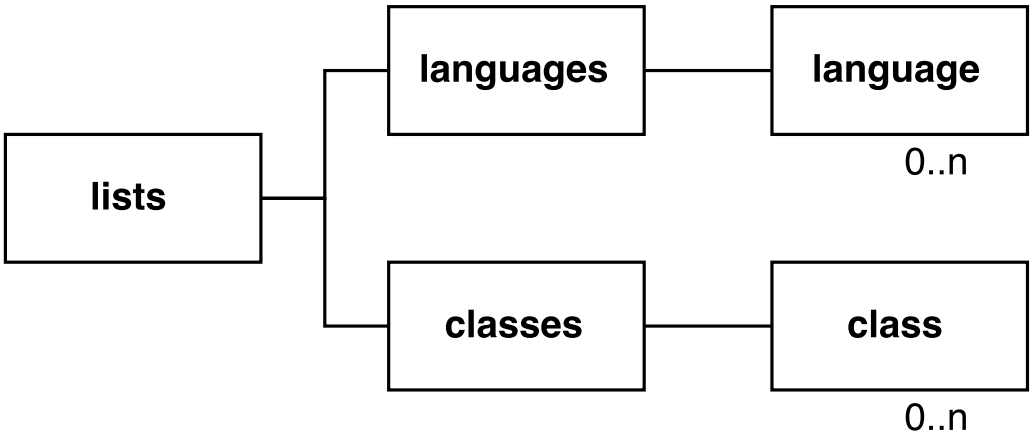
Tato příloha obsahuje návrh aplikace pro správu zdrojového kódu znázorněný pomocí UML diagramu tříd.


Main
«enumeration»
XmlKeyword
«abstract»
ADataManagementException
Config
is
XmlManagementException
DialogFactory
1
«interface»
IDataManager
1
use
use
«implements»
DialogKeyContainer
use
use
WindowController
XmlManager

C Diagram struktury XML souborů

Tato příloha obsahuje diagram znázorňující strukturu jednotlivých typů XML souborů pro ukládání dat aplikace pro správu zdrojového kódu.

Hlavní soubor



Soubor zdrojových kódů

D Uživatelská příručka aplikace

V adresáři `CodeManager/bin` kořenového adresáře přiloženého CD se nachází spustitelný soubor aplikace pro správu zdrojových kódů `CodeManager.jar`. Po jeho spuštění se otevře hlavní okno uživatelského rozhraní aplikace v režimu zobrazování zdrojového kódu.

D.1 Popis uživatelského rozhraní

V levé části hlavního okna se nachází seznam uložených tříd v jazyce Java, pro které byly implementovány knihovny v ostatních uložených programovacích jazycích, s tlačítky pro přidání, úpravu nebo odstranění třídy. V horní části se nachází nalevo (nad seznamem) vyhledávací pole uložených tříd, napravo pak nabídka uložených jazyků, vedle níž se nacházejí tlačítka pro přidání, úpravu nebo odstranění jazyka. Pod nabídkou jazyků (v pravé části okna) je umístěno textové pole pro zobrazení a úpravu zdrojových kódů a pod ním tlačítka pro kopírování a vkládání kódu a spouštění a ukládání jeho úpravy. V horní části okna nad vyhledávacím polem a nabídkou jazyků se nachází hlavní menu aplikace, které sdružuje funkcionalitu tlačítek v okně.

D.2 Zobrazení kódu

Kliknutím na položku v seznamu tříd v levé dolní části okna dojde k jejímu označení. Výběr položky lze provést i zadáním počátečních písmen názvu (nezáleží na jejich velikosti) do textového pole v levé horní části okna a kliknutím na tlačítko `Najít`. V nabídce v pravé horní části okna lze vybrat programovací jazyk. Jakmile jsou třída i jazyka vybrány, v textovém poli v pravé dolní části okna se zobrazí uložený zdrojový kód knihovny představující označenou třídu ve zvoleném jazyce. Dokud se okno nachází v režimu zobrazení kódu, obsah textového pole je možné pouze prohlížet, označovat jeho části a kopírovat do schránky (pro zkopírování celého obsahu najednou slouží tlačítko `Kopírovat kód`). Příklad okna aplikace po načtení kódu v režimu zobrazení ukazuje obrázek D.1.

D.3 Změna v seznamu tříd

V seznamu Java tříd je možné přidávat, upravovat a mazat položky. Přidání položky lze spustit kliknutím na tlačítko `Přidat třídu`. Zobrazí se dialogové okno, do kterého je nutné zadat název nové třídy. Kliknutím na tlačítko `OK` v dialogovém okně se potvrdí přidání nové třídy se zadaným názvem do seznamu uložených tříd a je možné následně uložit zdrojové kódy pro tuto třídu. Pro úpravu označené položky slouží tlačítko `Upravit třídu`. Následně zobrazené dialogové okno slouží k zadání nového názvu třídy. Po kliknutí na tlačítko `OK` se změní název třídy v seznamu. Uložené zdrojové kódy pro původní název třídy zůstanou zachovány. K odstranění položky je určeno tlačítko `Odstranit třídu`. Po kliknutí na toto tlačítko se nejprve zobrazí potvrzovací dialog. Potvrzením pomocí tlačítka `OK` dojde k odstranění označené třídy ze seznamu. Uložené zdrojové kódy pro tuto třídu ve všech jazycích budou nenávratně odstraněny.

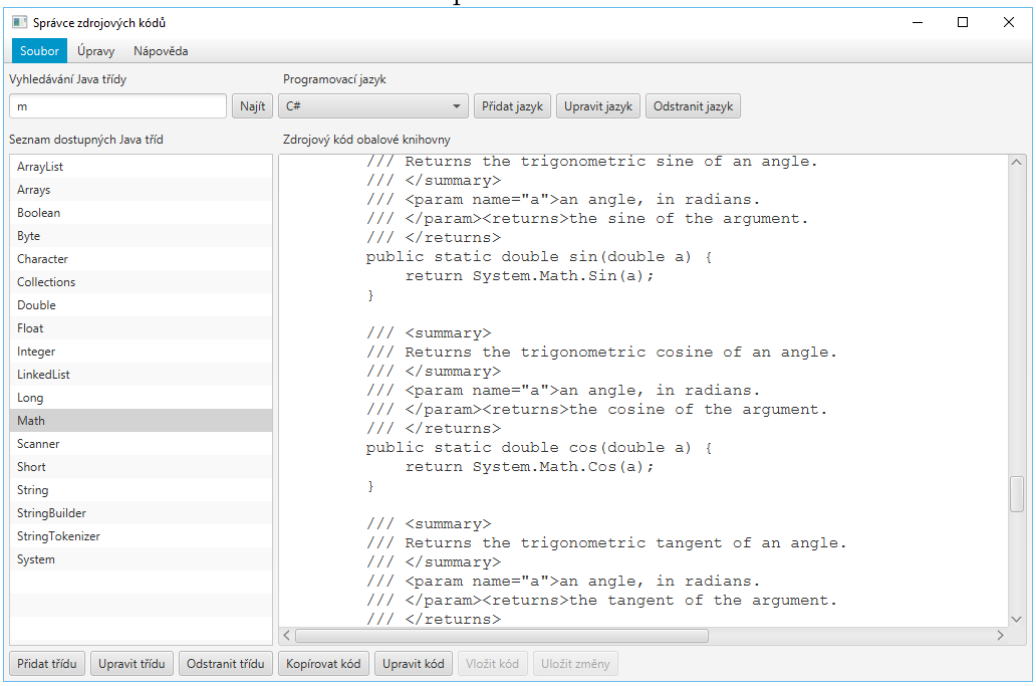
Pro zadávaný název třídy (ať už se jedná o vytvoření nové třídy nebo změnu názvu existující) platí pravidlo, že musí být validním názvem třídy v jazyce Java. Může tedy obsahovat pouze číslice (nesmí však číslicí začínat), písmena (velká i malá) a podtržítko. Dále se název nesmí shodovat s názvem žádné aktuálně uložené třídy, přičemž při porovnávání názvů na shodu se nerozlišuje velikost písmen.

D.4 Změna v seznamu jazyků

Nabídku programovacích jazyků lze modifikovat stejnými způsoby, jako seznam tříd. Stiskem tlačtka `Přidat jazyk`, zadáním názvu nového jazyka v dialogu a potvrzením tlačítkem `OK` přibude nový jazyk v nabídce. Data pro každou uloženou třídu je následně možné rozšířit o zdrojový kód v tomto nově uloženém jazyce. Stisknutím tlačítka `Upravit jazyk`, zadáním nového názvu a potvrzením tlačítkem `OK` se změní název jazyku v nabídce na nově zadaný. Zdrojové kódy z předchozího jazyka však zůstanou v každé třídě zachovány. Po stisku tlačítka `Odstranit jazyk` a potvrzení tlačítkem `OK` se odstraní zvolený jazyk z nabídky uložených jazyků. Uložené zdrojové kódy v tomto jazyce pro každou třídu budou nenávratně odstraněny.

Pravidla pro formát názvu programovacího jazyka jsou méně přísná než v případě názvů tříd. Název jazyka pouze nesmí začínat a končit mezerou, nesmí obsahovat více po sobě jdoucích mezer a nesmí obsahovat jiné bílé znaky než mezery. Stejně jako pro názvy tříd i zde platí, že se název nesmí shodovat s názvem aktuálně již uložené položky.

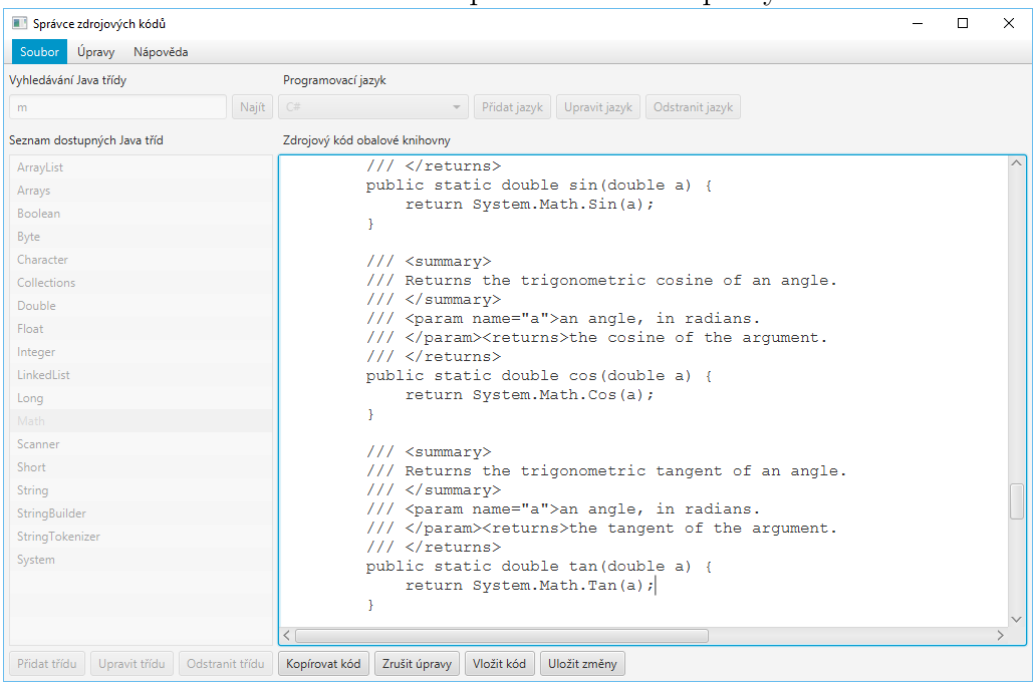
Obrázek D.1: Okno aplikace – režim zobrazení kódu



D.5 Úprava kódu

Pokud je vybrána Java třída i programovací jazyk, lze příslušný zdrojový kód upravit pomocí tlačítka `Upravit kód`. Po stisknutí tlačítka se rozhraní okna přepne do režimu úpravy kódu, jak ukazuje obrázek D.2. V tomto režimu není možné přepnout na jiný jazyk nebo třídu, je však možné editovat zdrojový kód zobrazený do textového pole v pravé dolní části okna (celý obsah lze poté tlačítkem Vložit změny také najednou nahradit obsahem schránky). Nová podoba kódu se uloží po kliknutí na tlačítko `Uložit změny`. Provedené změny v kódu je možné zahodit stisknutím tlačítka `Zrušit úpravy` a potvrzením zrušení pomocí tlačítka `OK` v dialogovém okně (stejný potvrzovací dialog se v tomto režimu zobrazí i při pokusu o ukončení aplikace). V takovém případě se načte zpět původní podoba kódu a provedené změny jsou ztraceny. Bezprostředně po uložení nebo zrušení změn se rozhraní okna přepne zpátky do režimu prohlížení kódu, kde je možné přepínat položky v seznamu tříd nebo v nabídce jazyků.

Obrázek D.2: Okno aplikace – režim úpravy kódu



E Nástroj pro převod kódu

K usnadnění převodu kódu psaném v jazyce Java do jiných jazyků byl vytvořen jednoduchý nástroj s webovým rozhraním. Soubor `index.html` pro jeho otevření v okně webového prohlížeče se nachází v adresáři pojmenovaném `CodeConverter`, nacházejícím se v kořenovém adresáři přiloženého CD. Ukázka webového rozhraní nástroje je na obrázku E.1

E.1 Popis uživatelského rozhraní

V horní části webového rozhraní pod popisem nástroje se nachází přepínače pro nastavení programovacího jazyka, do kterého bude kód převáděn. Pod nimi se nachází dvě textová pole. Pole v levé části je editovatelné a je určeno k zadání původního zdrojového kódu jazyka Java. Pole v pravé části slouží k zobrazení výsledku a je určeno pouze pro čtení.

E.2 Převod kódu

Vkládáním kódu do vstupního textového pole je automaticky spouštěn převod a jeho výsledky jsou vypisovány do výstupního textového pole. Nástroj pro správnou funkci vyžaduje v prohlížeči spuštěnou podporu jazyka JavaScript, ve kterém je psán převodní skript. Nástroj neprovádí komplexní převod a neprodukuje konečnou podobu kódu, poskytuje však možnost automatizace jednoduchých úkonů, jako je například přepis pojmenování datových typů a některých dalších klíčových slov.

E.3 Konfigurace

Skript pro převod kódu se nachází ve složce scripts v adresáři nástroje. Pro převody kódu používá soustavu regulárních výrazů sdružených s výslednými řetězci v jednoduchých objektech (atribut `p` obsahuje regulární výraz představující vzor nahrazovaných řetězců, atribut `r` obsahuje řetězec, kterým jsou výskyty vzoru nahrazovány), které jsou ukládány do polí pro každý programovací jazyk zvlášť. Funkcionalitu skriptu je možné jednoduše rozšířit přidáním dalších objektů s regulárními výrazy do příslušných polí, která jsou umístěna na začátek skriptu.

Obrázek E.1: Nástroj automatického převodu kódu