PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Vývojové diagramy

2013 Adam Londa

Anotace

Vývojové diagramy jsou grafy, které znázorňují činnost algoritmu. Jsou tedy vhodné mimo jiné pro výuku algoritmizace a programování. Náplní této bakalářské práce je vytvoření programu pro práci s vývojovými diagramy, vhodného pro výuku programování na středních školách.

Poděkování patří vedoucímu práce Mgr. Jiřímu Zacpalovi, Ph.D. za trpělivost, shovívavost, vstřícnost a za věcné připomínky při tvorbě programu i při psaní tohoto textu.
Díky patří také Tomáši Mikulovi za testování aplikace na Mac OS X.

Obsah

Seznam obrázků

1. Zadání práce

Program je určen pro výuku algoritmizace a programování na středních školách. Měl by umožňovat pohodlné vytváření a editaci algoritmů pomocí vývojových diagramů. Program by měl umět algoritmy spouštět, krokovat a převádět do kódu jazyka C. Dalším požadavkem je export diagramů do grafického formátu. Program by měl splňovat tyto požadavky:

- Pohodlné vytváření algoritmů pomocí vývojových diagramů – Samotná tvorba diagramů bude řešena pomocí Drag & Drop canvasu. Bude k dispozici paleta uzlů diagramu, ze které se budou jednotlivé prvky přidávat na canvas. Uzly budou vykresleny podle normy ČSN ISO 5807, uživatelské rozhraní bude v češtině, a bude obsahovat standartní strukturované menu.
- Spouštění a krokování algoritmů – Bude třeba navrhnout vnitřní reprezentaci jednotlivých typů uzlů nezávislou na uživatelském rozhraní, a implementovat obsluhu provádění znázorněných algoritmů. S tímto je spojena i nutnost ověření „spustitelnosti“ algoritmu. Průběh spuštěného algoritmu bude zobrazen na vývojovém diagramu. Součástí programu bude i „monitor“ zobrazující stav proměnných.
- Ukládání diagramů do XML a jejich zpětné načítání – Uložení stavu vnitřní reprezentace uzlů. Při načítání kontrola platnosti dat.
- Export diagramů do obrázku – Zachycení stavu canvasu, uložení do obrázku.
- Převod diagramů do zdrojového kódu jazyka C – Vnitřní reprezentace uzlů bude mít implementovaný převod do jazyka C. Výsledný kód algoritmu bude možno uložit na disk.
- Bude implementována multiplatformní desktopová aplikace splňující všechny výše uvedené body. Jazyk uživatelského rozhraní bude uložen odděleně, aby bylo možné vytvářet jazykové mutace programu. Stejně tak bude oddělena i norma pro kreslení vývojových diagramů.

2. Vývojové diagramy

Vývojový diagram je symbolickým jazykem používaným pro názorné zobrazení algoritmu či obecného procesu. Ke znázornění jednotlivých kroků se používají symboly propojené pomocí šipek znázorňujících tok řízení programu. [Wik01] Jde tedy o druh orientovaného grafu.

Vývojové diagramy mají velké uplatnění v informatice. Používají se při programovaní, pro návrh, analýzu, nebo pro dokumentaci. Velmi vhodné jsou ale i pro výuku programování a algoritmizace, protože umožňují názorně demonstrovat řešení problémů.

Pro kreslení vývojových diagramů existuje česká státní norma ČSN ISO 5807. Tato norma definuje symboly používané ve vývojových diagramech, které mohou reprezentovat program, systém, či tok dat. Státní norma definuje vývojový diagram programu jako zobrazení posloupnosti operací daného programu, skládající se ze symbolů pro jednotlivé operace, spojnic indikujících tok programu a speciálních symbolů pro usnadnění čtení a zápisu vývojového diagramu. [Kva01] Tato definice odpovídá obecnému popisu vývojových diagramů.

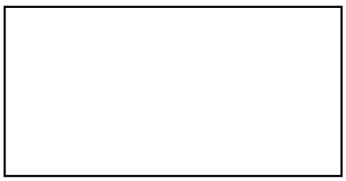
2.1. Symboly vývojových diagramů

Symboly vývojových diagramů jsou grafické značky s přesně definovaným významem. Pro upřesnění funkce se do symbolů umísťují slovní či symbolické popisy. Způsob psaní těchto popisů není normou určen, doporučuje se však používat jednoduchý text a ve výpočetních vztazích používat konvenční matematické symboly. Diagram tak bude srozumitelný bez ohledu na znalost konkrétního programovacího jazyka. [Kva01]

Následuje představení symbolů použitých v této bakalářské práci, odpovídající normě ČSN ISO 5807. Veškeré obrázky použité ke znázornění jednotlivých symbolů a texty popisující jejich význam jsou převzaté ze zdroje [Kva01].

2.1.1. Zpracování

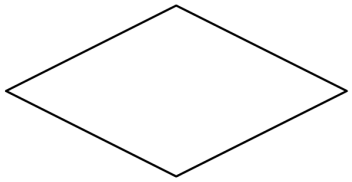
Symbol, který představuje obecně jakýkoliv druh zpracování, provedení definované operace nebo skupiny operací, jejichž výsledkem je transformování informace. (V případě této bakalářské práce je to změna hodnoty proměnné.) Tento symbol může mít obecně více vstupů, ale vždy jen jeden výstup.



Obrázek 1. Symbol zpracování

2.1.2. Rozhodování

Symbol představující rozhodovací nebo přepínací funkci. Symbol má jeden vstup a alternativní výstupy, z nichž je jeden zvolen po vyhodnocení podmínky uvnitř symbolu. Tento symbol má obvykle dva výstupy, ale obecně může mít i tři výstupy, nebo jeden výstup větvený.



Obrázek 2. Symbol rozhodování

2.1.3. Příprava

Symbol reprezentující obecně modifikaci činnosti, která mění postup činnosti následující. V případě této bakalářské práce se symbol používá pro reprezentaci cyklů. Symbol má dva vstupy. Jeden sekvenční, druhý pro návrat ze smyčky. Dále má symbol dva výstupy, jeden vstupující do smyčky a druhý sekvenční, který pokračuje ve výkonu programu.

Obrázek 3. Symbol přípravy

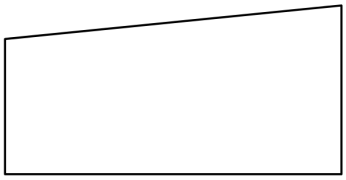
2.1.4. Vstup a výstup dat

Symbol představující obecně vstupně – výstupní operace s daty. Konkrétní druh vstupního či výstupního zařízení může být jednoznačně určen charakterem úlohy, může být popsán slovně, nebo je možné připojit symbol reprezentující vstupně – výstupní zařízení nebo nosič dat. Krom možnosti připojení vstupně – výstupního symbolu má tento symbol jeden vstup a jeden výstup, podobně jako symbol zpracování.

Obrázek 4. Symbol pro vstup a výstup

2.1.5. Ruční vstup

Symbol reprezentující nosiče a zařízení pro ruční vstup informací. V této bakalářské práci použit konkrétně pro vstup z klávesnice. Symbol má jen jeden výstup.



Obrázek 5. Symbol ručního vstupu

2.1.6. Zobrazení

Symbol, který představuje zařízení pro vizuální zobrazení dat, v případě této bakalářské práce reprezentuje výstup na monitor. Symbol má jen jeden vstup.

Obrázek 6. Symbol zobrazení

2.1.7. Mezní značka

Symbol představující obecně buďto vstup z vnějšího prostředí do programu, nebo výstup z programu do vnějšího prostředí. V této bakalářské práci použit pro indikaci začátku a konce programu. Symbol může mít buď pouze jeden vstup, nebo pouze jeden výstup.

Obrázek 7. Mezní značka

2.2. Spojnice

Spojnice je symbol v podobě svislé nebo vodorovné čáry představující řízení programu, sloužící ke spojení jednotlivých symbolů ve vývojovém diagramu. Standartní

směr toku informací je shora dolů a zleva doprava. Pokud je standartní tok dodržen, není třeba značit směr šipkami, avšak značení plnou nebo otevřenou šipkou se pro přehlednost doporučuje i přesto, že je standartní směr toku dodržen.

Spojnice je možné dle potřeby spojovat, avšak křížení spojnic se kvůli přehlednosti nedoporučuje. Dojde-li ke zkřížení, neznamená to žádný vzájemný vztah a žádné změny ve směru informací.

3. Existující aplikace

Součástí této bakalářské práce je i průzkum existujících aplikací pracujících s vývojovými diagramy, a jejich zhodnocení z hlediska výuky.

Průzkum každé z aplikací sestával ze sestrojení vývojového diagramu, pokusu o nalezení co nejvíce z funkcí, které jsou požadovány na program v rámci této bakalářské práce, a jejich případné odzkoušení.

Vedoucím práce, doktorem Jiřím Zacpalem, mi byly k otestování doporučeny programy *Yenka* a *Projekt Animované vývojové diagramy*. Mimo tato dvě řešení jsem nalezl program *SmartDraw* a webovou službu *LucidChart.com*.

3.1. Yenka

Yenka je program určený pro interaktivní výuku matematiky, fyziky, a informatiky. Pro studenty je dostupný zdarma, a je multiplatformní. Práce s vývojovými diagramy je v tomto programu řešena pomocí canvasu, na který se přetahováním přidávají uzly diagramu. Program umí algoritmy spouštět a pozastavovat. Možnost krokování zde chybí, ale stav proměnných lze při běhu algoritmu sledovat, takže krokování není třeba. Do zobrazovaných vývojových diagramů je možno zakreslovat také dílčí funkce, což studentům pomáhá osvojit si princip rozdělení problému na menší podproblémy. Vytvořené diagramy lze „vytisknout“ do PDF či do PostScriptu.

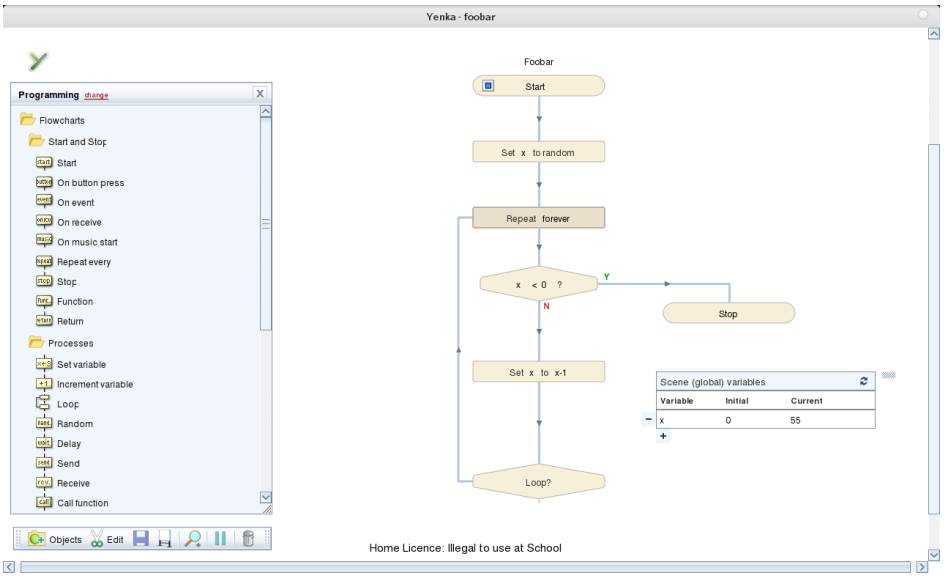
Uživatelské rozhraní Yenky je na první pohled poněkud nestandartní a nepřehledné, a zvyk na práci s programem zřejmě určitou dobu potrvá. Program je určen spíše pro mírně pokročilé studenty, a počítá se zde slespoň se základními znalostmi programování a algoritmizace.

3.2. LucidChart.com

Další z existujících řešení je webová služba *LucidChart.com*. Díky tomu, že se jedná o webovou aplikaci, je zde možné pracovat bez ohledu na platformu. S webovým řešením je ovšem spojena nutnost připojení k internetu, a v případě této aplikace zřejmě i potřeba moderního webového prohlížeče. Služba je s jistými (pro výuku nepodstatnými) omezeními dostupná zdarma.

Vývojové diagramy lze v této aplikaci tvořit pomocí Drag & Drop canvasu. Uživatelské rozhraní této služby je celkově velmi zdařilé, a uživatelsky přívětivé. Pokud je uživatel přihlášen pod svým účtem, který si zdarma zaregistroval, může si svou rozdělanou práci ukládat do cloudu. Nechybí zde ani možnost stažení diagramu v JPG, PNG, PDF, a mnoha dalších formátech. Co zde naopak chybí, je možnost spouštění a krokování zobrazovaných algoritmů.

Tato jinak velmi sympatická aplikace má pro výuku jednu velkou nevýhodu. A tou je sdílení vývojových diagramů. Lze sdílet jak s ostatními uživateli služby (v rámci neplaceného účtu omezeno na jednoho uživatele), tak i na sociálních sítích jako Facebook či Twitter, čímž se tato aplikace stává nepoužitelnou pro samostatné práce. I přes tento nedostatek je však toto řešení zajímavé, a rozhodně stojí za pozornost.



Obrázek 8. Yenka

3.3. SmartDraw

SmartDraw je komerční software určený pro profesionální tvorbu mnoha druhů grafů a diagramů, především pro vývojáře a manažery. Součástí tohoto rozsáhlého softwaru je i nástroj pro tvorbu vývojových diagramů. Jedná se o klasický canvas, do kterého se přidávají prvky z palety nástrojů. Uživatelské rozhraní tohoto programu využívá prostředí Ribbon, podobně jako například Microsoft Office. Diagramy lze exportovat do PDF, není zde ale možnost spouštění ani krokování znázorňovaného algoritmu.

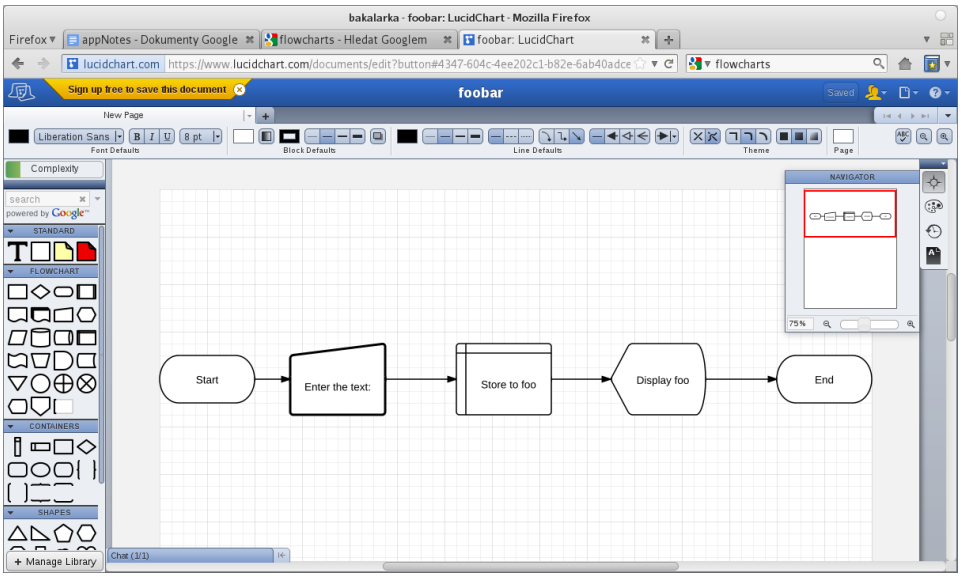
Tento program je pro výuku na středních školách obecně spíše nevhodný. Ať již z důvodu pořizovací ceny, platformové závislosti, chybějící možnosti názorného spouštění algoritmu, nebo kvůli uživatelskému rozhraní Ribbon, které je koncepčně určeno spíše pro profesionální nasazení osm a více hodin denně, nežli pro studenty, kteří jej používají pouze několik hodin týdně. Může však být vhodný pro pedagogy, jež chtějí studenty seznámit s prací (nejen) v softwarovém managementu.

3.4. Projekt Animované vývojové diagramy

Tento projekt Ing. Ivany Durdilové a Ing. Lenky Suchánkové ze SPŠE V Úžlabině v Praze je navržený speciálně pro výuku programování a algoritmizace na středních školách. Program obsahuje velké množství interaktivních, typicky středoškolských příkladů, i se slovním popisem činnosti. Algoritmy lze zobrazovat, spouštět a krokovat. Program také umožňuje tvorbu vlastních diagramů a export existujícího diagramu do bitmapového obrázku.

Animace znázorňující kroky algoritmu jsou názorné a přehledné. Je možné zadávat vstupní hodnoty, pokud je to součástí algoritmu. Jestliže pomineme (v tomto případě zanedbatelné) chyby v uživatelském rozhraní, jako například zbytečně časté používání silné zpětné vazby, nebo poněkud krkolomné zadávání vstupních hodnot, je tato část programu zvládnuta na výbornou, a pro výuku na školách je naprosto ideální.

Co je však již horší, je tvorba vlastních vývojových diagramů. Přidávat uzly lze pouze pomocí tlačítek, při přidávání startovního symbolu je nutné ručně zadat souřadnice. Chybí zde Drag & Drop, jednotlivé prvky na canvasu nelze vybírat, nelze



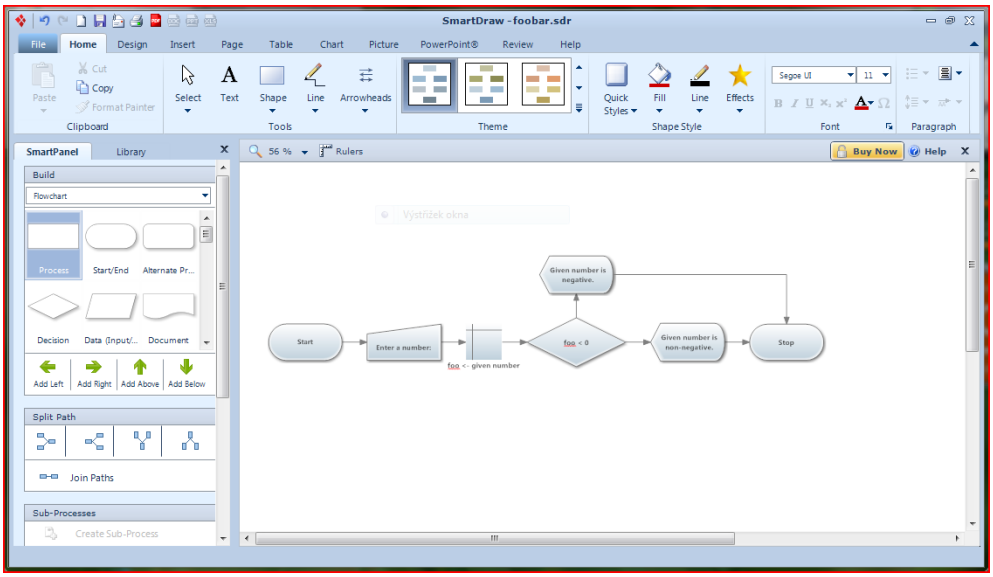
Obrázek 9. LucidChart.com

měnit velikost, ani jejich polohu. U symbolu rozhodování zde musíme předem znát délky obou větví. Celkově je zde tvorba vlastních vývojových diagramů uživatelsky nepřívětivá. Navíc chybí možnost znovuprovedení vráceného kroku, což je v tomto případě obzvláště nepříjemné. Vlastní diagramy lze ukládat, a opět načítat.

I přes výše zmíněné nedostatky patří tento projekt k tomu nejlepšímu, co lze na českých středních školách použít pro výuku programování. Program je určen pro Windows, ale bez větších problémů běží i na emulátorech jako například Wine. Pro výuku lze jednoznačně doporučit.

3.5. Závěr průzkumu existujících řešení

Průzkum existujících řešení ukázal, že žádná ze zkoumaných aplikací nesplňuje všechny požadavky, které jsou kladeny na program v rámci této bakalářské práce, a které odpovídají potřebám výuky na středních školách. Tento závěr průzkumu tedy budiž motivací pro implementaci aplikace splňující zadání této bakalářské práce.



Obrázek 10. SmartDraw

4. Implementace programu JFlowChart

4.1. Použité technologie

K implementaci programu JFlowChart jsem zvolil jazyk Java, především kvůli multiplatformnímu běhovému prostředí, a výbavě knihoven vhodné k vývoji moderních aplikací. Program je postaven na běhovém prostředí Java SE 1.7. Pro návrh uživatelského rozhraní jsem se rozhodl použít knihovnu SWT, kvůli „nativnímu“ vzhledu na všech platformách. K samotné implementaci jsem použil vývojové prostředí Eclipse. Program byl vyvíjen převážně na Linuxu, ale je otestován i na Windows a Mac OS X.

4.1.1. Java SE

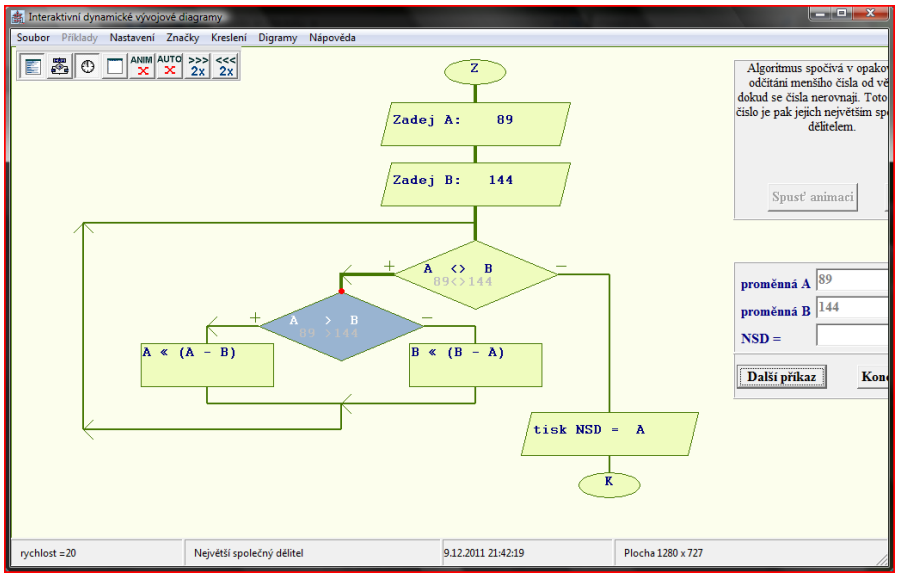
Java SE (neboli Java Standart Edition) je nejstarší platformou jazyka Java [Wik02] používaná pro desktopové počítače. [Wik03]

Jazyk Java je objektově orientovaný jazyk umožňující psát multiplatformní aplikace. Platformová nezávislost aplikací je zajištěna překladem do tzv. *bajtkódu*, který je při běhu aplikace zpracováván tzv. *virtuálním strojem Javy*. Virtuální stroj Javy je součástí běhového prostředí Java, které je třeba pro běh aplikací psaných v Javě nainstalovat.

4.1.2. SWT

Standart Widget Toolkit neboli SWT je alternativní knihovna grafických uživatelských prvků pro platformu Java. K vykreslování GUI prvků používá tento toolkit nativní knihovny operačních systémů prostřednictvím Java Native Interface.[1] Programy používající SWT jsou přenosné, ale implementace toolkitu, ačkoliv je napsán v Javě, je unikátní pro každou platformu. [Wik04]

[1] Rozhraní umožňující propojit kód běžící na virtuálním stroji Javy s nativními programy a knihovnami napsanými v jiných jazycích [Wik05]



Obrázek 11. Projekt Animované vývojové diagramy

4.2. Vnitřní struktura programu

Vnitřní strukturu programu zapouzdřuje balík `engine`. Ten je pro přehlednost rozdělen do několika dalších podbalíků:

- Balík `engine.arithmetics` obsahuje implementaci proměnných, hodnot a operací, které jsou obsaženy v některých typech uzlů.
  - Rozhraní `Printable` – Obecně vše, co lze zobrazit jako řetězec znaků, uložit do XML, provádět copy/paste.
  - Rozhraní `Operand` – Šablona pro operandy. Rozšiřuje rozhraní `Printable`.
  - Rozhraní `Operation` – Předpis operace. Lze provádět hloubkovou kopii a vyhodnocovat pomocí metody `Object evaluate()`. Opět rozšiřuje rozhraní `Printable`.
  - Abstraktní třída `Value` – Implementace hodnoty, číselné i nečíselné, používané jako operand.
  - Rozhraní `Variable` obecně popisuje proměnnou, rozšíření rozhraní `Operand`. Zajišťuje nastavení hodnoty metodou `setValue(Value v)` a možnost zjištění názvu a datového typu proměnné.
  - Abstraktní třída `GeneralVariable` – Obecná proměnná dle rozhraní `Variable`. Obsahuje kód společný pro všechny implementované proměnné.
  - Rozhraní `ComparableOperand` je popis operandu, který lze porovnat s jiným porovnatelným operandem metodou `int compareTo(ComparableOperand o)` vracející rozdíl operandů, stejně jako stejnojmenná nativní javovská funkce.
  - Třída `ComparableValue` – Implementace hodnoty použitelné jako porovnatelný operand.
  - Třída `ComparableVariable` reprezentuje proměnnou coby porovnatelný operand.

  - Rozhraní `NumberOperand` popisuje číselný operand jako speciální případ porovnatelného operandu. Přidává sčítání, odčítání, násobení a dělení dvou číselných operandů.
  - Třídy `NumberValue` a `NumberVariable` jsou rozšířením tříd `ComparableValue` a `ComparableVariable` o číselné operace dle rozhraní `NumberOperand`.
  - Abstraktní třída `BinaryOperation` – Reprezentace obecné binární operace.
  - Abstraktní třída `ComparableBinaryOperation` – Pomocná „mezitřída“, umožňující použít binární operaci jako porovnatelný operand.
  - Třída `NumberExpression` – Implementace binární číselné operace, použitelné jako porovnatelný i vnořený číselný operand.
  - Výčtový typ `ConditionValue` – Tříhodnotová logika sestávající z hodnot `TRUE`, `FALSE` a `UNKNOWN`, používaná u podmínek při testu spustitelnosti diagramu.
  - Třída `Comparison` – Binární operace porovnání. Po vyhodnocení vrací hodnotu `true` nebo `false`, oba operandy porovnatelné. Implementován také test na kontradikci.
  - Třída `Assignment` – Binární operace přiřazení. Na levé straně vždy proměnná, na pravé straně libovolný operand.
  - `IncompatibleDataTypesException` a `InvalidOperatorException` jsou výjimky sloužící k signalizaci chybových stavů při vytváření a vyhodnocování operací a výrazů.

- Balík `engine.nodes` obsahuje vnitřní reprezentaci uzlů a spojovacích bodů vývojového diagramu.
  - Základem je rozhraní `Node`, předepisující jak metody pro práci s informacemi o uzlu (souřadnice umístění či velikost uzlu, informace o výběru či nastavení breakpointu), tak i předpisy polymorfních metod pro copy/paste, aktivaci spojovacího bodu, sémantickou kontrolu proměnných, dosažitelnosti konce diagramu, kontrolu smyček, uložení do XML, generování zdrojového kódu, a výkon akce během průchodu uzlem.
  - Abstraktní třída `GeneralNode` implementující rozhraní `Node` obsahuje kód společný pro všechny druhy uzlů. Právě zde jsou uloženy informace o umístění, velikosti či spojovacích bodech uzlu.
  - Abstraktní „mezitřída“ `StartEndNode` obsahuje společný kód pro počáteční a koncový uzel grafu. Zavedena kvůli redundanci kódu.
  - Třídy `StartNode` a `EndNode` potom reprezentují samotný počáteční a koncový uzel.
  - `InputOutputNode` je další pomocná abstraktní třída, tentokrát zobecňující uzly pro vstup a výstup.
  - Třídy `DisplayNode` a `ManualInputNode` jsou implementace uzlů pro zobrazení a pro ruční zadání hodnoty proměnné.
  - Třída `ProcessNode` reprezentuje uzel, který dané proměnné přiřadí danou hodnotu.
  - Třída `DecisionNode` implementuje uzel rozhodování podle nastavené podmínky.

  - Třída `PreparationNode` – Implementace uzlu představující cyklus určený podmínkou a krokovou operací. Tento uzel je klíčový pro polymorfní metodu detekující nekonečný cyklus.
  - Třida `ConnectPoint` je implementací spojovacích bodů, které obsahuje každý uzel. Instance této třídy obsahují informace o umístění bodu, zda je bod vstupní či výstupní, o uzlu, kterému bod patří, obsahuje také seznam spojovacích bodů, se kterými je propojen. (Obecně jich může být více, například bod, do kterého se „sbíhají“ větve programu.) Prostřednictvím propojovacích bodů je realizováno volání polymorfních metod mezi propojenými uzly. Každý spojovací bod má také unikátní ID, používané při ukládání či načítání plátna.
  - Třída `AssistantConnectPoint` je rozšířením třídy `ConnectPoint` implementující pomocný spojovací bod. Pomocné spojovací body nepatří k žádnému uzlu, a jsou vždy výstupní.
  - Dalším rozšířením třídy `ConnectPoint` je třída `LoopConnectPoint`, navržena speciálně pro cykly. Přepsáním metod pro komunikaci mezi propojenými uzly je tak zabráněno vzniku zacyklení. Použití ve třídě `PreparationNode`.
  - Třída `Connection` zapouzdřuje informace o propojení dvou spojovacích bodů. Kromě dvou bodů „odkud – kam“ obsahuje i seznam pomocných spojovacích bodů, pokud přes ně spojení vede. Tato třída se používá při záznamu kroků undo/redo a také při načítání plátna z XML.

- `engine.history` je balík obsahující implementace záznamů akcí pro undo/redo.
  - Rozhraní `Action` obecně popisuje akci pro zásobník undo/redo. Předepisuje metody `undo()`, `redo()` a metodu `recordNewData()` používanou při záznamu editace.
  - Abstraktní třída `GeneralAction` implementující rozhraní `Action` obsahuje informaci o plátně, na kterém se změny provádí, společné pro všechny druhy akcí.
  - Třídy `AddingAction` a `RemovingAction` zaznamenávají přidání a odstranění jednoho nebo více uzlů z plátna.
  - Třídy `EditingAssignmentAction`, `EditingConditionAction`, `EditingPreparationAction` a `EditingInputOutputAction` jsou implementace akcí mapujících editaci dat pro různé druhy uzlů.
  - Třídy `AddingConnectionAction`, `EditingConnectionAction` a `RemovingConnectionAction` zase umožňují záznam manipulace se spojeními mezi uzly – přidávání, upravování a odebírání existujících spojení.
  - Třída `MovingAction` – Záznam přemístění uzlů z místa A na místo B.
  - Třída `ResizingAction` – Záznam změny velikosti plátna.

- Balík `engine.runtime` obsahuje implementaci sémantiky zobrazovaného grafu.
  - Třída `Program` je implementací sémantiky diagramu. Zajišťuje běh při spouštění a krokování diagramu, spouští kontrolu spustitelnosti a generování zdrojového kódu. Obsahuje seznam proměnných používaných v algoritmu, obsahuje referenci na instanci třídy `StartNode` určující začátek programu na plátně.

  - Výčtový typ `SemaphoreState` je souhrn stavů programu. `MISSING_START` pro chybějící začátek programu, `UNSET_VARIABLE` pro používání proměnné, která nebyla nastavena, `UNREACHABLE_END` značící nedosažitelný konec programu a `REACHABLE_END` znamenající dosažitelný konec a spustitelný program.
  - Třída `Semaphore` slouží jako indikátor stavu spustitelnosti grafu na plátně. K indikaci používá prvky výčtu `SemaphoreState`, a pro aktualizaci stavu spouští kontroly na nastavené instanci třídy `Program`.

- Balík `engine.xml` zajišťuje manipulaci s XML soubory při načítání a ukládání stavu plátna. Pro práci s XML soubory jsem zvolil knihovnu StAX, především kvůli rychlosti a přehlednosti kódu.

  - Třída `Labels` je knihovna názvů pro XML uzly, používaných jak při ukládání, tak při načítání.
  - Třída `XMLWriter` definuje objekt pro zápis stavu plátna do XML. K zápisu slouží metoda `save(OutputStream out, FlowChartCanvas c)`, která do předaného výstupního proudu zaznamená stav předaného plátna.
  - Třída `XMLReader` zajišťuje načítání uloženého stavu. Ke čtení slouží metoda `load(InputStream in, FlowChartCanvas c)`, která předanému plátnu nastaví obsah přečtený z předaného vstupního proudu.
  - `XMLReaderException` a `XMLWriterException` jsou výjimky signalizující chybové stavy při ukládání či načítání XML souboru.

- Balík `engine.codegen` obsahuje rozhraní `CodeGenerator` předepisující implementaci generátoru zdrojového kódu. V podbalíku `engine.codegen.c` je implementován generátor zdrojového kódu v jazyce C. Samotné generování probíhá prostřednictvím metody `String generate(Program p)`, která na předané instanci třídy `Program` spustí volání polymorfní metody u jednotlivých uzlů grafu. Výsledek navrátí v řetězci znaků.

- Balík `engine.common` obsahuje třídy používané v celém programu. Součástí tohoto balíku je knihovna konstant (třída `Const`), knihovna pomocných statických funkcí (třída `Functions`) a jazyk uživatelského rozhraní uložený ve třídě `Language`. Jde vlastně o knihovnu řetězců reprezentujících nejrůznější nápisy používané v celém programu. Řetězce jsou při incialazici programu nastaveny metodou nesoucí jméno daného jazyka (například `czech()` pro češtinu) a tuto instanci třídy `Language` si poté předávají všechny objekty, které jazyková data používají. Zatím implementována pouze čeština.

4.3. Implementace uživatelského rozhraní

Uživatelské rozhraní programu JFlowChart je implementováno v toolkitu SWT. Veškeré třídy reprezentující uživatelské rozhraní jsou uložené v balíku `ui`. Pro přehlednost jsou opět rozdělené do podbalíků:

- Balík `ui.canvas` obsahuje implementaci plátna – stěžejní části uživatelského rozhraní. Samotné plátno je implementovano ve třídě `FlowChartCanvas`. Balík dále obsahuje třídu `CanvasPlayer` zajišťující animaci plátna při spouštění zobrazovaného algoritmu, a třídu `CanvasEnvironment` poskytující zálohu stavu plátna před vyprázdněním a obnovení stavu v případě chyby při načítání z XML.
- Podbalík `ui.canvas.graphnodes` obsahuje implementaci vykreslení na plátno pro všech sedm druhů uzlů. Implementace je předepsána rozhraním `GraphNode` rozšiřující rozhraní `Node` z balíku `engine.nodes` o metodu `draw(GC gc)`.
- V balíku `ui.windows` jsou implementována veškerá okna a dialogy aplikace JFlowChart

4.4. Obrazová a textová data

Kromě balíků oddělujících vnitřní strukturu programu od uživatelského rozhraní obsahuje implementace programu JFlowChart i zdroje (především obrazových) dat. Jsou to balíky `icons`, `flowchartSymbols.csnIso5807` a ZIP archiv `help.zip`.

Balík `icons` obsahuje ikony používané v nástrojové liště. Balík `flowchartSymbols.csnIso5807` obsahuje PNG soubory se symboly vývojových diagramů[2] dle normy ČSN ISO 5807, které se používají při vykreslování uzlů na plátno. V archivu `help.zip` je obsažena záloha HTML souborů použitých v nápovědě. Pokud při inicializaci nápovědy program nenalezne adresář `help` v adresáři, ze kterého byl spuštěn, provede obnovení souborů z přibaleného ZIP archivu.

4.5. Externí knihovny

Program JFlowChart používá externí knihovny `swt.jar` a `swing2swt.jar`. Knihovna `swt.jar` obsahuje implementaci toolkitu SWT a knihovna `swing2swt.jar` umožňuje v SWT použití některých layoutů z nativního javovského GUI toolkitu Swing.

[2] Obrazová data převzata z [Kva01]

5. Představení programu JFlowChart

5.1. Požadavky ke spuštění

5.1.1. Podporované operační systémy

Použitá verze knihovny SWT podporuje následující platformy:

- Windows XP, Windows Vista a Windows 7 s grafickou knihovnou Win32
- AIX, FreeBSD, Linux, HP–UX a Solaris s knihovnou GTK+
- Mac OS X používající knihovnu Cocoa

Pro jiné než výše zmíněné platformy s uvedenými grafickými knihovnami není zaručen správný chod aplikace.

5.1.2. Verze Javy

K bezproblémovému běhu aplikace JFlowChart je třeba nainstalovat běhové prostředí Java verze 7 nebo vyšší.

Ke každé platformě je k dispozici 32bitová a 64bitová verze programu JFlowChart. Tyto verze se vztahují k nainstalovanému běhovému prostředí Javy, ne přímo k operačnímu systému.

5.2. Standartní spuštění programu

Nelze-li program spustit poklepáním na spustitelný JAR archiv, lze program spustit z příkazové řádky:

- **Windows & Linux:** `java -jar JFlowChart.jar`
- **Mac OS X:** `java -XstartOnFirstThread -jar JFlowChart.jar`

5.3. Průvodce programem

Hlavní okno programu JFlowChart sestává z několika částí.

V centrální části okna se nachází plátno, jehož prostřednictvím probíhá práce s diagramy.

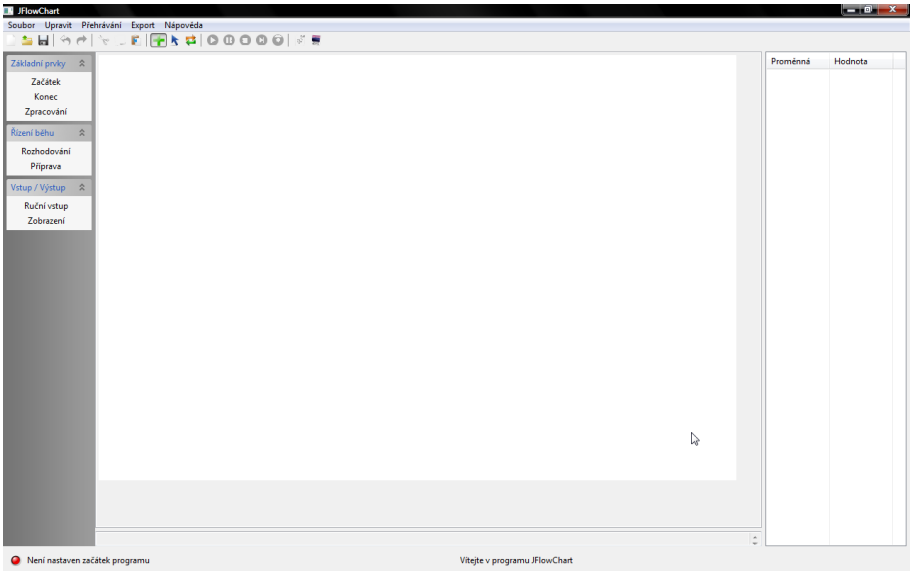
Nalevo od plátna je paleta uzlů, ze které vybíráme, když chceme na plátno přidat uzel.

Nad plátnem se nachází nástrojová lišta. Vedle standartních prvků pro vytvoření nového plátna, načítání, ukládání, undo/redo a práci se schránkou jsou umístěny přepínače mezi Režimem přidávání, Režimem výběru a Režimem spojování. Dále se na nástrojové liště nachází ovládací prvky pro spouštění a krokování diagramu a tlačítka pro export do zdrojového kódu v jazyce C a do obrázku PNG.

Napravo od plátna je tabulka, která při spuštěném diagramu monitoruje stav proměnných.

Pod plátnem se nachází konzole, na kterou se zobrazuje výstup, je-li součástí zobrazeného algoritmu.

V levé části stavového řádku naspodu okna je semafor, který informuje o stavu spustitelnosti diagramu na plátně. Pravá část stavové lišty slouží k zobrazení tipů a informací se slabou zpětnou vazbou.

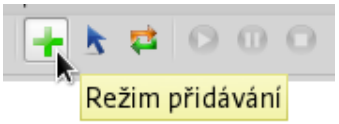


Obrázek 12. Hlavní okno programu JFlowChart

5.3.1. Režimy práce

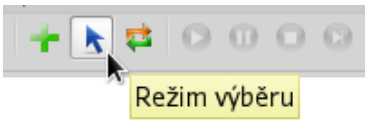
V programu JFlowChart existují tři režimy práce s uzly na plátně: Režim přidávání, režim výběru a režim spojování. Přepínače sloužící ke změně režimu se nacházejí na nástrojové liště nalevo od ovládacích prvků pro spouštění a krokování.

Režim přidávání umožňuje přidávat na plátno uzly. Na nástrojové liště je reprezentován zeleným symbolem plus.



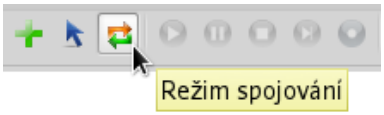
Obrázek 13. Režim přidávání

Režim výběru umožňuje vybrat jeden či více uzlů na plátně a s výběrem provádět například přesuny, úpravy operací, nebo mazání. V toolbaru je symbolizován modrým kurzorem.



Obrázek 14. Režim výběru

Režim spojování umožňuje veškerou manipulaci s propojením uzlů – vytváření, editaci, či odstranění existujících spojení. V panelu nástrojů symbolizován dvěma barevnými šipkami.



Obrázek 15. Režim spojování

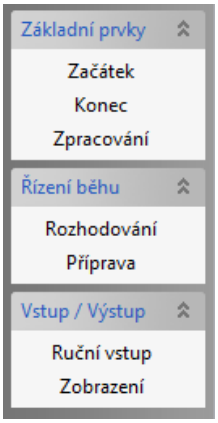
5.3.2. Uzly vývojového diagramu

Na paletě vlevo od plátna máme k dispozici uzly vývojového diagramu. Jednotlivé druhy uzlů[3] jsou rozděleny do třech skupin: Základní prvky, Řízení běhu a Vstup/Výstup.

Mezi Základní prvky patří mezní symboly označující začátek a konec vývojového diagramu. Dále do této skupiny spadá symbol Zpracování, reprezentující přiřazení hodnoty k proměnné, což je jeden ze dvou způsobů, jak do diagramu zavést proměnnou.

Do skupiny Řízení běhu patří symbol Rozhodování reprezentující větvení programu podle zadané podmínky, a symbol Přípravy představující cyklus určený podmínkou a krokovou operací.

Ve skupině Vstup/Výstup najdeme symbol Ručního vstupu, představující druhý způsob, jak v programu inicializovat proměnnou, a symbol zobrazení umožňující zobrazení hodnoty proměnné na výstup.



Obrázek 16. Paleta uzlů

5.3.3. Nastavení uzlů

U většiny uzlů, které jsou v programu JFlowChart k dispozici (s výjimkou mezních uzlů pro začátek a konec diagramu) je třeba nastavit proměnnou, podmínku, nebo operaci, kterou uzel provádí. K nastavení těchto vlastností slouží speciální dialogy.

U symbolu Zpracování je na levé straně vždy proměnná, na pravé straně potom hodnota, číselný výraz, nebo jiná proměnná kompatibilního datového typu.

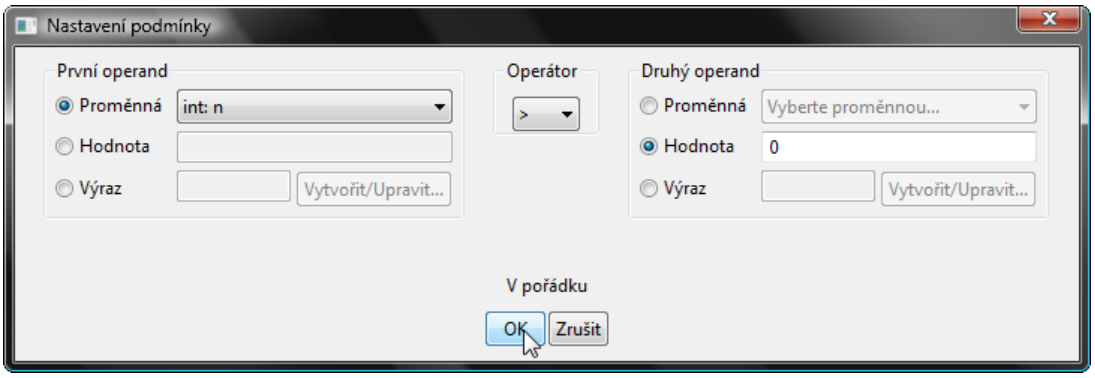
U podmínek (ať už v rámci symbolu Rozhodování či Přípravy) je možno na obou stranách nastavit proměnné, hodnoty nebo číselné výrazy. Pokud však používáme proměnné, musí už v okamžiku nastavení existovat. Podporované operace pro podmínky jsou: >, <, >=, <=, == a ! =.

[3] O významu jednotlivých druhů uzlů a jejich symbolech podrobněji v kapitole Vývojové diagramy

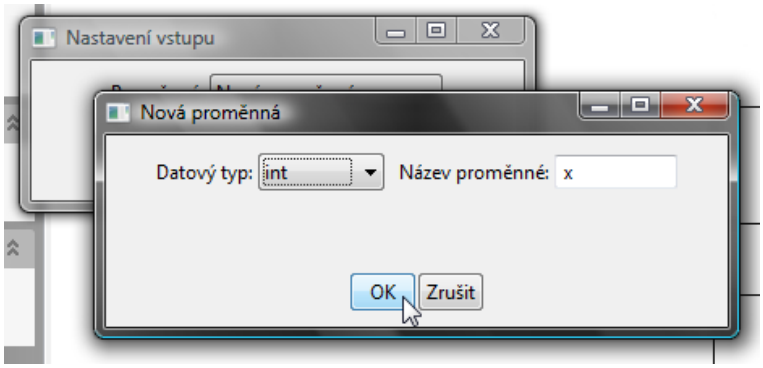
U nastavení číselných výrazů platí pro operandy stejná pravidla, jako u podmínek. (Proměnné musí při nastavování existovat a součástí číselného výrazu může být i vnořený číselný výraz.) Podporované operace pro číselné výrazy jsou plus, mínus, krát a děleno.

5.3.4. Přidávání, upravování a odebírání uzlů

V režimu přidávání lze přidávat uzly na plátno. Jsme-li v režimu přidávání, můžeme z palety uzlů vybrat prvek a klepnutím jej na plátno přidat. U většiny implementovaných uzlů (kromě mezních symbolů) se po klepnutí na plátno zobrazí dialogové okno pro nastavení činnosti uzlu. Součástí nastavení uzlu může být i vytváření nových proměnných nebo číselných výrazů.



Obrázek 17. Nastavení činnosti uzlu



Obrázek 18. Vytvoření nové proměnné

Chceme-li uzly na plátně odstranit, v režimu výběru vybereme jeden či více uzlů k odstranění a stiskem klávesy Delete nebo volbou *Odstranit* v kontextovém menu jednoho z vybraných uzlů provedeme odstranění. Klepnutím a tažením po plátně lze v režimu výběru vybírat plošně.

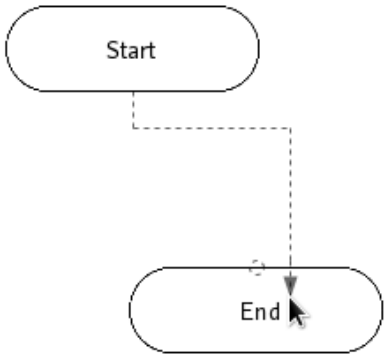
Pro upravení činnosti uzlu zvolíme v režimu výběru uzel k úpravě a v kontextovém menu vybraného uzlu zvolíme *Upravit uzel...* V režimu výběru možno upravit také poklepáním na uzel.

Uchopením a tažením vybraných uzlů lze uzly v režimu výběru přemísťovat.

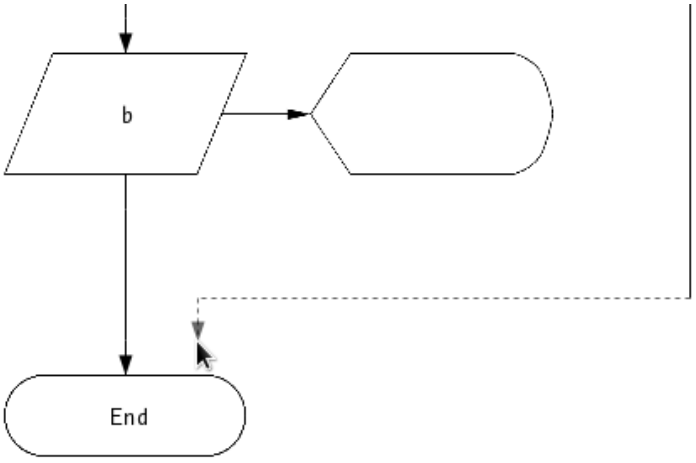
5.3.5. Spojování uzlů

V režimu spojování můžeme uzly na plátně spojovat. Spojovací bod se zvýrazní, pokud nad vybraný uzel najedeme myší. Chceme-li ze zvýrazněného bodu vést spojení, klepnutím myši spojování zahájíme. Spojování ze zvoleného uzlu lze zrušit dvojklikem, nebo stiskem klávesy Esc. Před zvolením cílového uzlu lze také klepnutím na plátno zavést pomocné spojovací body a ovlivnit tak cestu spojnice, což může být užitečné, chceme-li se vyhnout křížení. Při volbě cílového uzlu opět dojde ke zvýraznění spojovacího bodu. Poté lze klepnutím spojení dokončit.

Existující spojení lze upravit klepnutím na cílový uzel, nebo odstranit klepnutím na uzel, ze kterého spojení vede.



Obrázek 19. Spojování uzlů

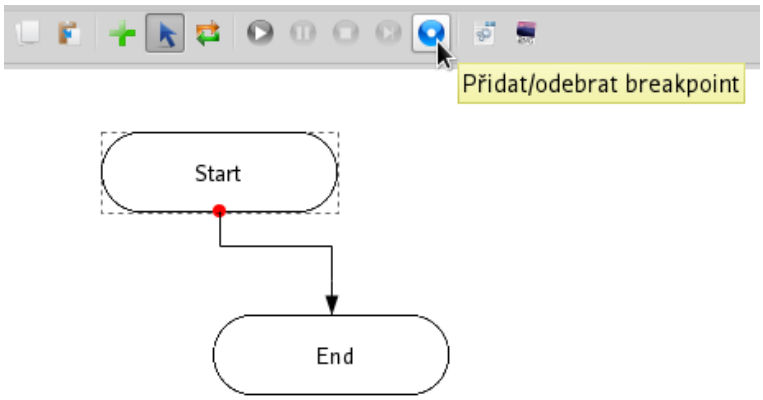


Obrázek 20. Spojování uzlů s využitím pomocných bodů

5.3.6. Spouštění a krokování diagramů

Svítí-li indikátor v levém dolním rohu okna zeleně, je možné algoritmus zobrazený na plátně spustit. K ovládání „playbacku“ slouží sada ovládacích prvků umístěných na liště nad plátnem, možno ovládat také z klávesnice. Pokud je přehrávání pozastaveno, je možné algoritmus krokovat. Krom možnosti „ručního“

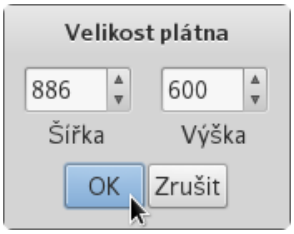
pozastavení je zde možnost přidání breakpointu. Ten lze přidat v režimu výběru tím, že vybereme uzel, ve kterém se má výpočet pozastavit a zvolíme možnost *Přidat/odebrat breakpoint*.



Obrázek 21. Ovládácí panel pro spouštění a krokování, přidání breakpointu

5.3.7. Změna velikosti plátna

V programu JFlowChart lze měnit velikost plátna. Jsou dvě možnosti, jak to provést. První možností je v režimu výběru uchopit a táhnout některý z okrajů plátna. Druhá možnost je upravit velikost skrze volbu *Velikost plátna...*, přičemž je možno v dialogovém okně zadat číselné hodnoty.



Obrázek 22. Dialog pro změnu velikosti plátna

6. Závěr a zhodnocení programu

6.1. Co se povedlo

Podařilo se implementovat multiplatformní aplikaci, která pohromadě poskytuje funkce potřebné pro výuku algoritmizace a programování na středních školách. Díky rozhraní SWT má aplikace přívětivý vzhled na všech platformách. Je implementováno pohodlné vytváření vývojových diagramů, přehrávání, krokování, export do obrázku. Obsažena je i možnost vygenerování zdrojového kódu, kterouž jsem u žádného ze zkoumaných existujících řešení nezaregistroval.

6.2. Co se nepovedlo

Jednou ze stinných stránek programu je potřeba přesunutí všech uzlů do viditelné části plátna pro export do obrázku. Toolkit SWT je schopen zachytit pouze viditelnou část plátna. Nenalezl jsem žádný vhodný způsob, jak prostřednictvím knihovny SWT do obrázku zachytit celé plátno. Další slabinou je konzole, sloužící jako výstup algoritmu se slabou zpětnou vazbou. Kvůli omezením layoutů SWT šlo umístit konzoli, která má velikost pouhého jednoho řádku, ačkoli je scrollovaná. Nepřišel jsem na způsob, jak konzoli zvětšit, aniž bych tím narušil zbytek rozhraní.

6.3. Možnosti rozšíření

Ačkoli program splňuje všechny body zadání, rozhodně je do budoucna co přidat a vylepšit. Například, implementace pole jako datového typu a iterování skrz prvky. Jistě by to usnadnilo modelování nejednoho problému. Implementace tvorby funkcí a jejich volání, například rozšířením o další mezní symbol. Může být implementována práce s více plátny, například v panelech. Přehrávání diagramů by mohlo mít nastavitelnou rychlost animace. Uživatelské rozhraní je zatím v češtině, může být přeloženo do dalších jazyků přidáním inicializační funkce do třídy `Language` v daném jazyce. Struktura jazykové knihovny může být přepsána do XML, pro lepší přehlednost. Třída `Language` by tak nepotřebovala inicializační funkce, ale mohla by jazyk načíst z XML. Snížila by se redundance kódu a rozšiřování o další jazyky by bylo snadnější. Je implementován generátor kódu v jazyce C, avšak předpis generátoru rozhraním umožňuje rozšíření o další jazyky, například Python či Pascal. Výtvarně nadaní jedinci mohou navrhnout intuitivnější symboly ikon. Implementace poznámek na plátně by byla pro potřeby výuky také jistě užitečná. Zajímavá by mohla být také možnost exportu do PDF či PostScriptu. Rozhraní `Operation` může být rozšířeno o unární operace, čímž by byla umožněna například implementace mocniny či odmocniny. Mohl by být také implementován parser pro převod řetězce na číselný výraz, což by umožnilo implementaci uživatelsky přívětivější tvorby číselných výrazů prostým vepsáním do formuláře. Do nastavení symbolu Výstupu může být přidána možnost nastavit formátovaný výstup. Jak je tedy vidno, nápadů pro další vývoj tohoto programu je stále více než dost.

Reference

[Kva01] Internet *http://www.ikvalita.cz/downloads/kap2.pdf*
Autor a rok neznámý

[Wik01] Internet *http://cs.wikipedia.org/wiki/Vývojový_diagram*
Wikipedia, 2013

[Wik02] Internet *http://cs.wikipedia.org/wiki/Java_SE*
Wikipedia, 2013

[Wik03] Internet *http://cs.wikipedia.org/wiki/Java_(programovací_jazyk)*
Wikipedia, 2013

[Wik04] Internet *http://cs.wikipedia.org/wiki/Standard_Widget_Toolkit*
Wikipedia, 2013

[Wik05] Internet *http://cs.wikipedia.org/wiki/Java_Native_Interface*
Wikipedia, 2013