STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

VÝUKOVÝ SOFTWARE – KIRCHHOFFOVY ZÁKONY

Jakub Hrnčíř

Liberec 2011

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor SOČ: 12. Tvorba učebních pomůcek, didaktická technologie

VÝUKOVÝ SOFTWARE – KIRCHHOFFOVY ZÁKONY

EDUCATIONAL SOFTWARE – KIRCHHOFF'S CIRCUIT LAWS

Autor: Jakub Hrnčíř

Škola: Gymnázium F. X. Šaldy,
Partyzánská 530, Liberec 11,
příspěvková organizace

Liberec 2011

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně, použil jsem pouze podklady uvedené v přiloženém seznamu a postup při zpracování a dalším nakládání s prací je v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Liberci dne: podpis:

Poděkování.

Děkuji Mgr. Janu Voženílkovi za podnětné připomínky, které mi během práce poskytoval.

ANOTACE

Cílem této práce je vytvoření aplikace pro výuku a procvičování výpočtu chování elektrických obvodů podle Kirchhoffových zákonů. Ty popisují obvody stejnosměrného elektrického proudu. Jejich aplikace má tu výhodu, že není matematicky složitá. Elektrické proudy se počítají pomocí soustav lineárních rovnic.

Aplikace Kirchhoff byla naprogramována v jazyce Java. Tato aplikace uvádí uživatele do problematiky a poskytuje prostředí, které vede uživatele od zakreslení obvodu až ke konečným výsledkům. Lze ji využít při samostudiu i pro zpestření školní výuky. Aplikace Kirchhoff je plně funkční, ale jsou zde samozřejmě možnosti rozšíření.

Klíčová slova: Kirchhoffovy zákony, výukový software, elektrický obvod, stejnosměrný proud, návrh obvodu

ANNOTATION

The aim of this work is to create an application for teaching and practice of calculating the behaviour of electrical circuits using Kirchhoff's circuit laws. They describe circuits of direct electric current. Their application has the advantage of not being mathematically difficult. Electric currents are calculated using systems of linear equations.

The application Kirchhoff was programmed in Java. It gives users the background information about the laws and provides an environment that guides the user from drawing the circuit to the final results. It can be used for self study and for school education. The application Kirchhoff is fully functional but it can be also extended.

Keywords: Kirchhoff circuit laws, educational software, electrical circuit, direct current, electrical circuit design

Obsah

Práce obsahuje také dalších 41 stran příloh.

1 Úvod

Hlavním cílem této práce je vytvoření aplikace pro výuku a procvičování výpočtu chování elektrických obvodů podle Kirchhoffových zákonů. Aplikace obsahuje stručný úvod do problematiky a poskytuje prostředí, které krok za krokem vede uživatele od zakreslení obvodu až ke konečným výsledkům. Lze ji využít při samostudiu i pro zpestření školní výuky.

Kirchhoffovy zákony popisují chování elektrických obvodů stejnosměrného elektrického proudu. Jejich aplikace má tu výhodu, že není matematicky složitá. Elektrické proudy se počítají pomocí soustav lineárních rovnic.

2 Teorie Kirchhoffových zákonů

Text této kapitoly je dostupný přímo z aplikace a slouží jako stručný výukový text určený k pochopení základů této látky.

2.1 Základní pojmy použité v aplikaci

Obvod – soustava vodičů, uzlů a komponent zobrazená na mřížce

Uzel – místo styku tří a více vodičů 

Komponenta – zdroj stejnosměrného proudu 

– rezistor (odpor) 

Větev – část obvodu, který leží mezi dvěma uzly (vodič i s komponentami)

Smyčka – uzavřená dráha tvořená některými větvemi obvodu

Prvek obvodu – uzel, větev nebo smyčka

2.2 Kirchhoffovy zákony

2.2.1 První Kirchhoffův zákon

Je to v podstatě zákon zachování náboje formulovaný pro uzel. Proud je definován jako náboj, který proteče průřezem vodiče za jednu sekundu. Tento náboj se nemůže v uzlu ztrácet, ani hromadit. Proto platí:

Součet proudů vcházejících do uzlu roven součtu proudů z uzlu vycházejících.

Znění tohoto zákona může vypadat i takto: *Algebraický součet všech proudů v uzlu je nulový.*

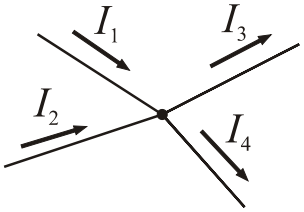
$$\sum_{k=1}^{n} I_k = 0$$

V této formulaci je třeba zavést znaménkovou konvenci. Obvykle mají proudy do uzlu přitékající kladná znaménka, proudy z uzlu vytékající mají znaménka záporná.

2.2.2 Příklad

Rovnice pro uzel na obrázku vypadá takto:

$I_1 + I_2 - I_3 - I_4 = 0$



2.2.3 Druhý Kirchhoffův zákon

Součet úbytků napětí na spotřebičích je roven součtu elektromotorických napětí zdrojů.

$$\sum_{k=1}^{n} R_k I_k = \sum_{l=1}^{n} U_l$$

Při psaní rovnic podle druhého Kirchhoffova zákona je třeba dodržet tento postup:

1. Vyznačíme zvolené směry proudů ve všech větvích. Zvlášť vyznačíme směr postupu smyčkou.
2. Úbytky napětí (součin odporu spotřebiče a proudu jím protékajícího) zapisujeme jako kladné, pokud souhlasí směr postupu smyčkou se směrem očekávaného proudu ve větvi (a jako záporné, pokud nesouhlasí).
3. Napětí zdrojů zapisujeme jako kladné, pokud směr postupu smyčkou vstupuje do zdroje záporným pólem (a jako záporné, pokud kladným).

Výsledek je nezávislý na zvolených směrech proudů a postupů. Pokud se při označování směrů proudů nezvolí správný směr, vyjde velikost proudu záporná.

2.2.4 Počet potřebných rovnic

Pro každý obvod lze vytvořit větší počet rovnic. Některé z nich ale jsou jen součty či rozdíly jiných a je tedy zbytečné s nimi počítat. Celkem je třeba tolik rovnic, kolik je neznámých proudů (tzn. kolik je větví v obvodu).

Pro rovnice prvního Kirchhoffova zákona platí, že je vždy potřeba o jednu rovnici méně, než je počet uzlů.

Rovnic druhého Kirchhoffova zákona (a tedy i nezávislých smyček) je potřeba:

$V - U + 1$ (V je počet větví, U počet uzlů). Pro rovnice druhého Kirchhoffova zákona je potřeba vybírat nezávislé smyčky. Daný počet smyček je třeba rozdělit tak, aby každá větev byla součástí alespoň jedné smyčky. Pokud jsou dodržena tato pravidla, měly by být smyčky nezávislé.

2.3 Další informace

Pokud se zajímáte o toto téma, či potřebujete znát větší podrobnosti, doporučuji učební text z knihovničky fyzikální olympiády, který naleznete na následující adrese:

http://fo.cuni.cz/texty/elobvody.pdf

3 Metodika

Při vývoji aplikace byl nejdříve navržen způsob, jakým uživatel určí podobu počítaného obvodu. Byla zvolena a implementována metoda zakreslení do čtvercové sítě pomocí myši. Poté bylo naprogramováno převádění vstupního obvodu z mřížky do logické struktury. Logická struktura obvodu je nezbytná pro automatické vytvoření rovnic. Dále bylo pomocí externí knihovny Apache Commons Math[1] implementováno vyřešení systému lineárních rovnic.

Potom byly vytvořeny části aplikace, které postupně vedou uživatele od návrhu obvodu, přes vytvoření rovnic až k jejich vyřešení a interpretaci jejich výsledků. Nakonec byla vytvořena nápověda a dokument uvádějící uživatele do problematiky (viz kapitola 2).

3.1 Použité technologie

3.1.1 Java SE

Java je objektově orientovaný programovací jazyk vyvinutý firmou Sun Microsystems, která byla v roce 2009 koupena firmou Oracle.

Objektově orientované programování (zkráceně OOP) je způsob programování, kdy se řešený problém rozdělí na objekty, které mají definované vlastnosti a metody, které je možno na nich provádět. Objekty stejných vlastností jsou instancemi jedné třídy (Class).

Java je interpretovaný jazyk, to znamená, že programátor vytváří tzv. mezikód, který je nezávislý na operačním systému a typu zařízení. Aplikace potom funguje na každém zařízení, které má nainstalováno interpret Javy, tzv. virtuální stroj (JVM – Java Virtual Machine), je to tedy aplikace multiplatformní.

Pro vývoj aplikace jsem použil vývojové prostředí NetBeans IDE.

3.1.2 HTML

HTML (z anglického HyperText Markup Language, hypertextový značkovací jazyk) je jazyk pro vytváření stránek v systému World Wide Web, který umožňuje publikaci stránek na Internetu. Je charakterizován množinou značek a atributů, které slouží převážně k formátování obsahu stránky. V této aplikaci byl použit pro formátování textů (nápověd, popisů prvků atd.).

[1] Více informací a zdrojové kódy knihovny na http://commons.apache.org/math/

3.1.3 SVN

Při vývoji jsem použil systém Apache Subversion (zkráceně SVN), který umožňuje správu verzí zdrojového kódu. V případě chyby ve vývoji pak umožňuje návrat ke starším verzím zdrojového kódu bez jinak zdlouhavého opravování.

K tomuto účelu jsem využil software TortoiseSVN. Tento SVN klient umožňuje využívat technologii SVN pod Microsoft Windows a je zdarma.

3.2 Některé řešené problémy s implementací

3.2.1 Java Web Start

Software Java Web Start (JWS) umožňuje spustit Java aplikace jednoduchým kliknutím na odkaz v prohlížeči bez zbytečně složitého stahování a instalace aplikace. Usnadněním přístupu se zvyšuje potenciální využití aplikace, ale pro tvůrce, který s JWS nemá zkušenosti, přináší spoustu problémů.

Odkaz spouštějící aplikaci ukazuje na Java Network Launching Protocol (.jnlp) soubor, který zajistí automatické stažení a spuštění aplikace. Bylo však obtížné nalézt správnou podobu JNLP souboru, aby spuštění fungovalo, protože nesrozumitelné chybové hlášky Javy vůbec nepomáhaly v odstranění jejich příčin.

Dále bylo třeba nalézt správný způsob připojení zdrojových souborů aplikace, protože JWS kvůli bezpečnosti uživatele poskytuje jen velmi omezené možnosti přístupu k souborům. Správným způsobem nakonec bylo přibalení zdrojových dat přímo do JAR souboru prostřednictvím úpravy souboru build.xml, který obsahuje informace jak celou aplikaci sestavit ze zdrojových kódů.

3.2.2 Externí matematická knihovna Apache Commons Math

Výhodou použití externí knihovny k řešení soustav lineárních rovnic bylo ušetření práce. Ale i tato cesta měla své obtíže. Použitá knihovna prochází neustálým vývojem a zdokonalováním, o něco hůře je na tom její dokumentace. Našel jsem návod, jak pomocí knihovny řešit daný problém, ale nebyl aktuální. Některé třídy a funkce od doby napsání návodu změnily svá jména nebo strukturu. Nakonec jsem problém vyřešil vyhledáním ekvivalentního postupu přímou inspekcí zdrojových kódů knihovny.

4 Popis aplikace

4.1 Šiřitelnost

Aplikace je bezplatně přístupná na internetové adrese http://kirchhoff.gfxs.cz. Uživatel ji může buď otevřít přímo přes prohlížeč pomocí Java Network Launching Protocol, nebo stáhnout JAR soubor, pro jehož spuštění musí mít uživatel nainstalováno Java Runtime Environment[2].

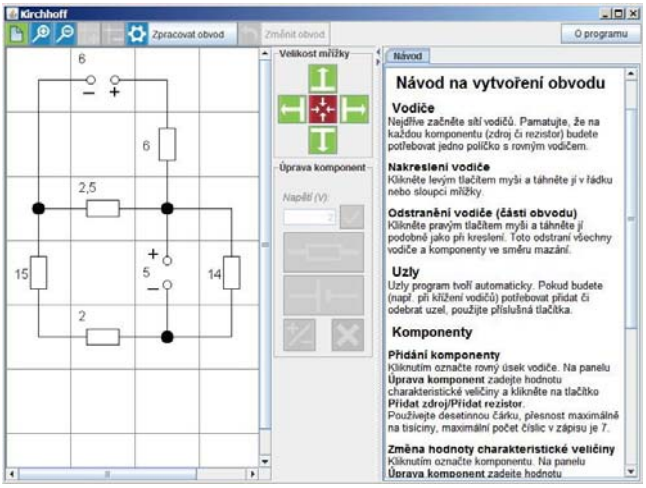
4.2 Způsob použití ve výuce

Aplikace je vytvořena tak, aby mohl uživatel látku pochopit sám. Je tedy určena hlavně pro mimoškolní výuku a procvičování, ale lze použít i jako zpestření výuky či jako pomůcka pro rychlé řešení základních úloh na Kirchhoffovy zákony.

Pro školní výuku s pomocí této aplikace je ideální, pokud může probíhat v počítačové učebně. Každý student tak může individuelně používat aplikaci pro řešení úloh.

4.3 Stručný popis aplikace

Aplikace se mění při procházení jednotlivých fází řešení úlohy, základní rozvržení se však nemění. V horním řádku jsou tlačítka pro ovládání postupu mezi fázemi a způsobu zobrazení obvodu, v levé části je mřížka s obvodem. V pravé části jsou informace pro uživatele, jako je nápověda pro danou fázi řešení nebo rovnice a jejich řešení. Mezi mřížkou a obvodem jsou v první fázi nástroje na úpravu obvodu, v dalších fázích informace o jednotlivých prvcích obvodu. Poměr místa určený pro informace a pro nákres lze ovládat tažením levé hrany informačního panelu. Informační panel lze také maximalizovat nebo dočasně skrýt.



Obrázek 1 – Základní vzhled aplikace

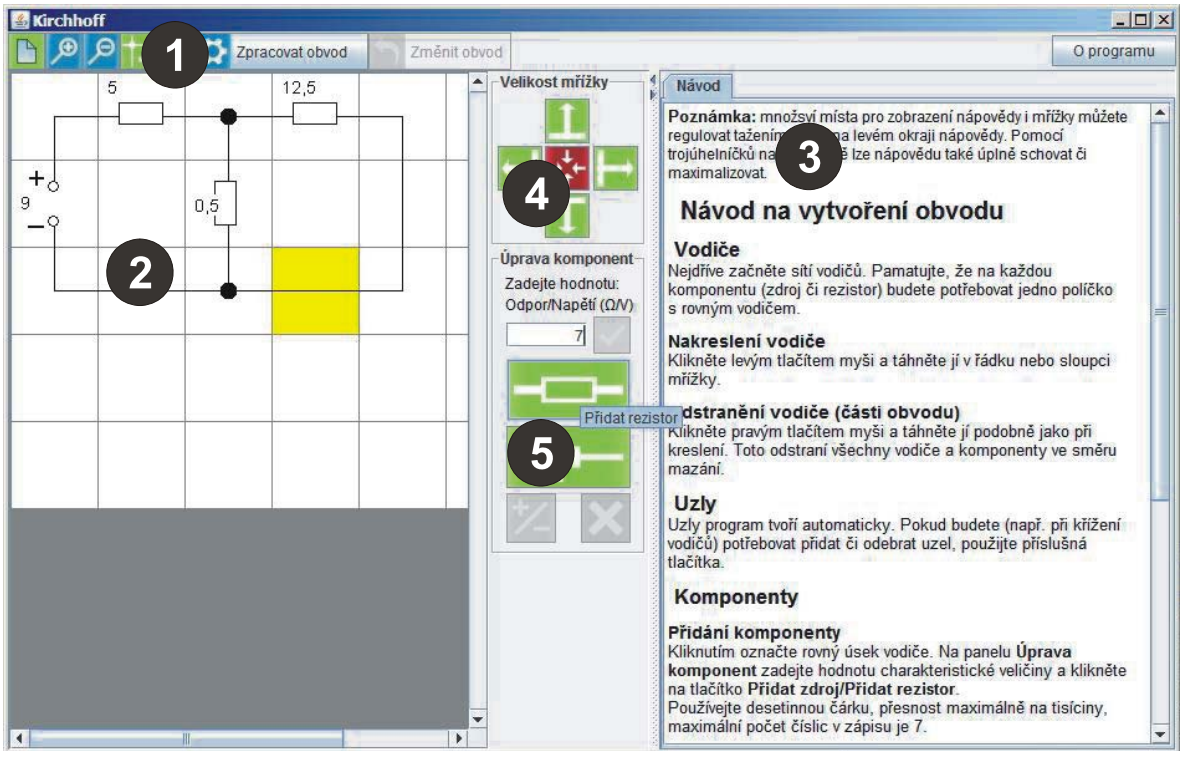
[2] dostupné zdarma na internetu – např.: http://www.java.com/download/

5 Průvodce ovládáním aplikace

5.1 Návrh obvodu

Nejprve je třeba vytvořit obvod. Náhled aplikace je na obr. 2. Uživatel buď vymyslí vlastní obvod, nebo do aplikace překreslí již zadaný obvod (např. z učebnice). Při kreslení obvodu se uživatel řídí podrobnou nápovědou vpravo. Obvod je kreslen pro přehlednost do čtvercové sítě. Čtvercová síť má nevýhodu, že do ní nelze zakreslit uzel, kde se stýkají více než 4 vodiče. Ve většině úloh se ale tato situace nevyskytuje, případně ji lze uskutečnit postupným větvením s více uzly za sebou.

Když má uživatel obvod hotov, stiskne tlačítko **Zpracovat obvod**. Program zadaný obvod vyhodnotí, a pokud nalezne nedostatky zabraňující provedení výpočtu, upozorní na ně uživatele a nedovolí mu pokračovat, dokud je neodstraní. Typickým problém může být například zkratování obvodu nebo výskyt více nezávislých obvodů.

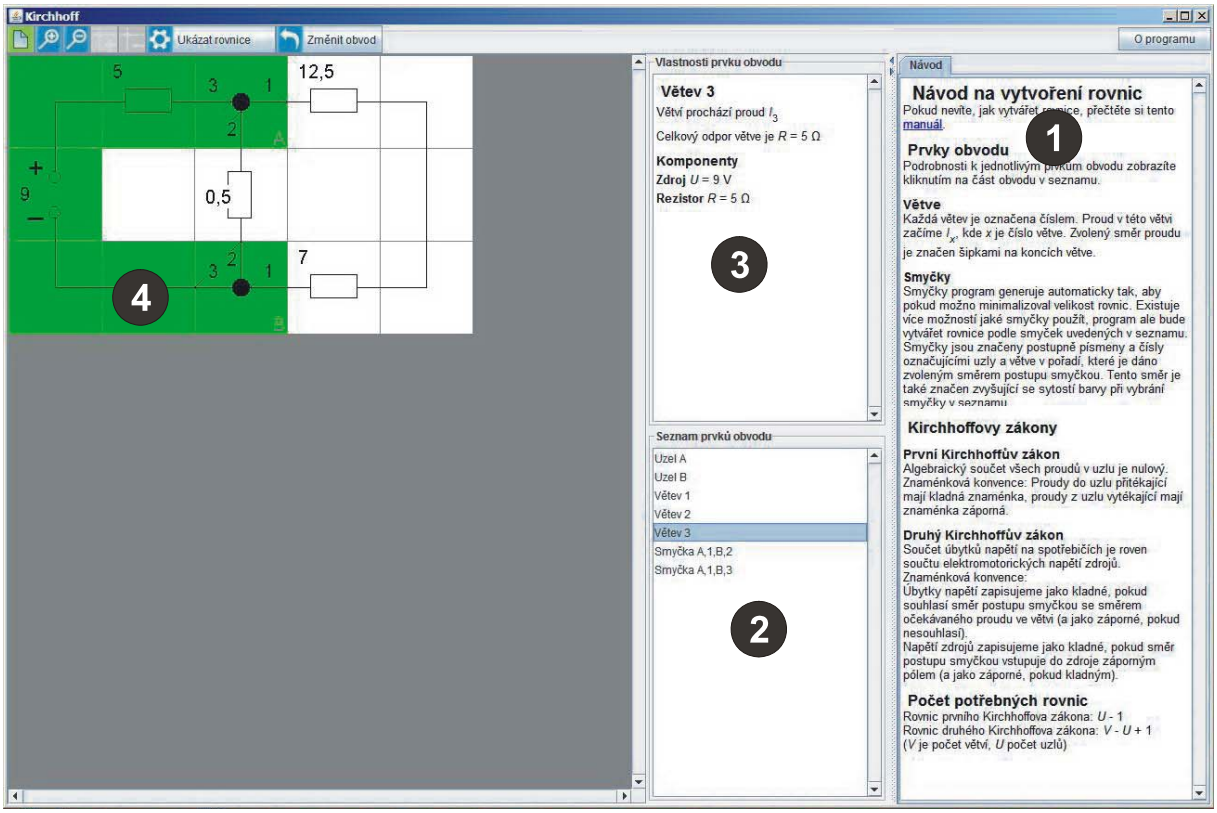


Obrázek 2 – Návrh obvodu

1. Horní panel – přesun mezi fázemi, nová mřížka, zvětšení/zmenšení mřížky
2. Mřížka s obvodem – žluté vybarvení pole znamená, že je toto pole vybráno pro editaci
3. Informační panel – v první fázi obsahuje kompletní návod na vytvoření obvodu.
4. Nástroj Velikost mřížky – umožňuje rozšiřovat mřížku do 4 směrů a oříznutí prázdných okrajů
5. Nástroj Úprava komponent – umožňuje přidávat a odebírat komponenty a měnit jejich vlastnosti

5.2 Tvorba rovnic

V této fázi (obr. 3) je na uživateli, aby si na papíře (v sešitě) vytvořil rovnice pro tento obvod. Postup na jejich vytvoření je popsán v návodu. Uživatel může využít také popisů a vyznačení jednotlivých prvků obvodu. Aplikace zvolí směry proudů ve větvích a vybere smyčky, ze kterých je třeba vytvořit rovnice. Pokud má uživatel rovnice hotovy, nebo si neví rady, stiskne tlačítko **Ukázat rovnice**.



Obrázek 3 – Tvorba rovnic

1. Stručná nápověda k vytvoření rovnic (Podrobná verze je dostupná přes odkaz v nápovědě – viz kapitola 2)
2. Seznam prvků – seznam uzlů, větví a smyček obvodu
3. Vlastnosti prvku – po vybrání prvku ze seznamu se zobrazí jeho popis, je-li k dispozici
4. Po vybrání prvku ze seznamu se prvek vyznačí zelenou barvou v obvodu

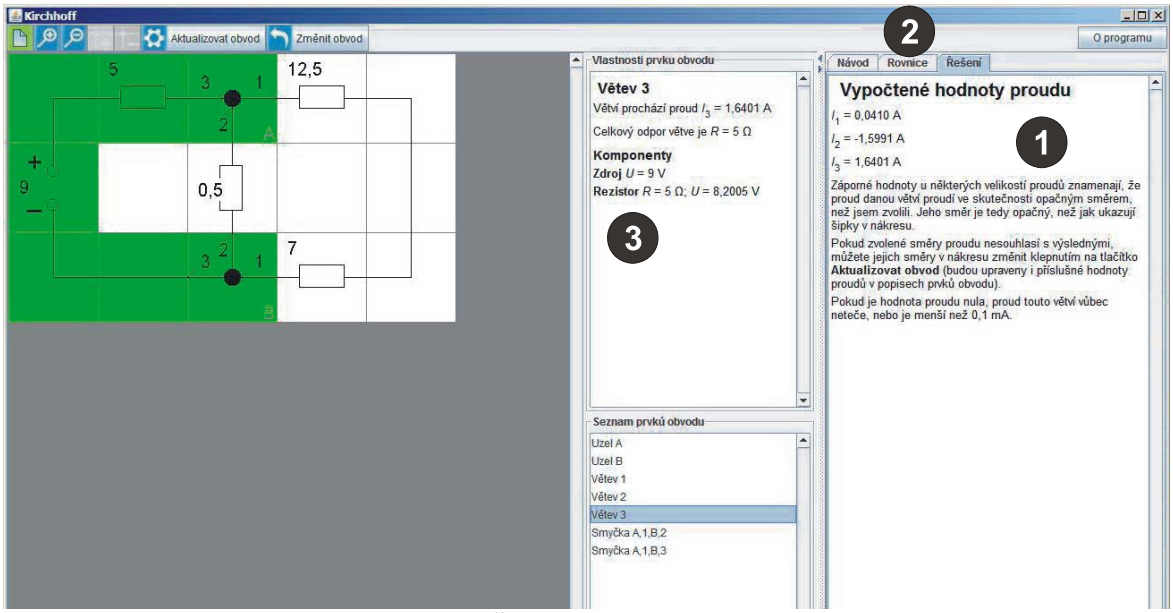
5.3 Výpočet rovnic

V této části si uživatel zkontroluje rovnice (obr. 4), jestli je vytvořil správně. Pokud má nějakou nejasnost, v záložce Návod se stále nachází návod na jejich vytvoření, který by společně s výslednými rovnicemi měl vést k pochopení systému. Je na uživateli, zdali rovnice spočítá (z důvodů procvičení doporučeno), nebo rovnou přejde stiskem tlačítka **Spočítat rovnice** k jejich řešením.

Obrázek 4 – Vytvořené rovnice zobrazené v informačním panelu

5.4 Interpretace řešení

V této fázi jsou uživateli zobrazena numerická řešení rovnic (obr. 5). Pokud vyšly všechny proudy jako kladné, odpovídají určené směry proudu v nákresu směrům skutečným a úloha je vyřešena. Ve většině případů se to ale nestává a řešení je třeba správně interpretovat. Společně s řešením rovnic se zobrazí i pokyny k jejich interpretaci. Stisknutím tlačítka **Aktualizovat obvod** se změní směry proudů a popisy tak, aby odpovídaly vypočteným a správně interpretovaným výsledkům.

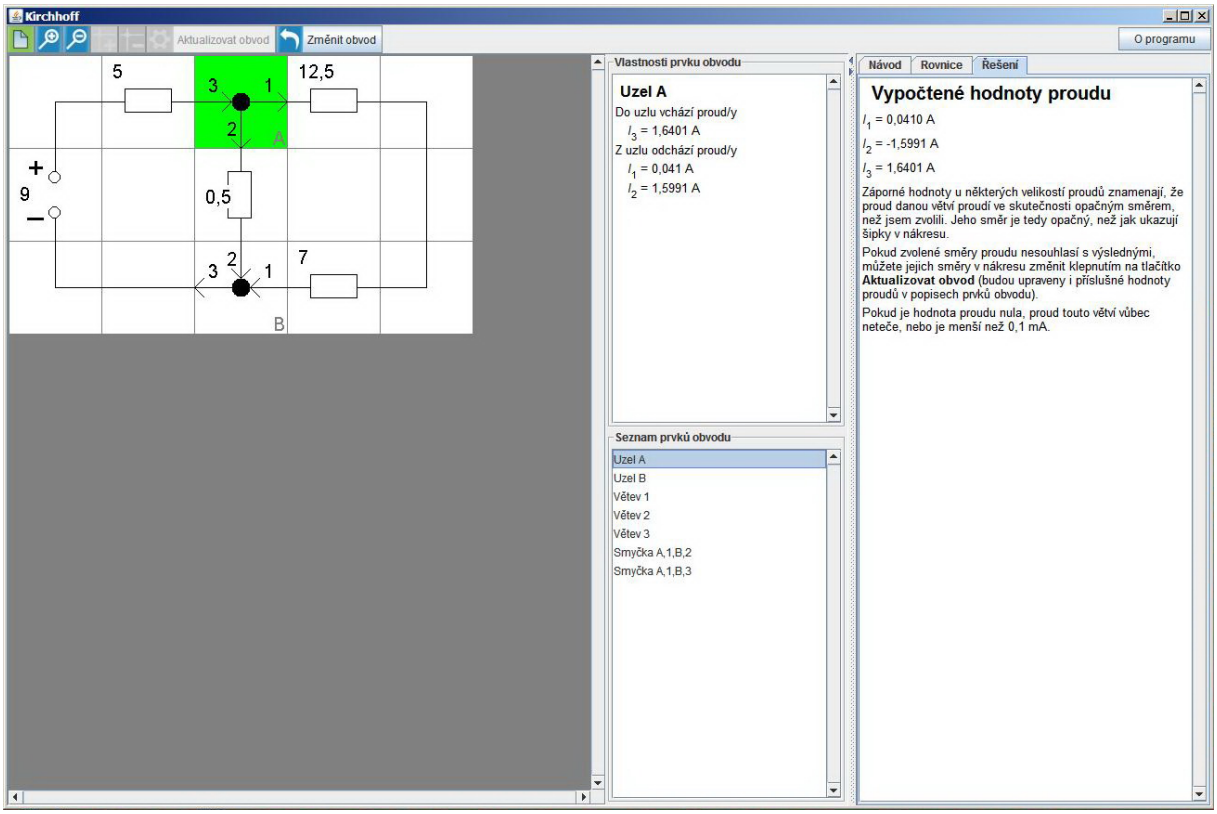


Obrázek 5 – Řešení rovnic a jejich interpretace
Vysvětlivky následují na další straně

1. Výsledná numerická řešení a pokyny k jejich interpretaci
2. Původní rovnice jsou stále k dispozici v záložce Rovnice
3. Popis prvků je doplněn o vypočtené hodnoty.

5.5 Konečný výsledek

V této fázi (obr. 6) je úloha hotova, uživatel vidí nákres obvodu se správnými směry proudu. V popisech prvků obvodu si může prohlédnout hodnoty proudů a napětí v jednotlivých větvích.



Obrázek 6 – Konečný výsledek – obvod se správnými směry proudů a popisy prvků

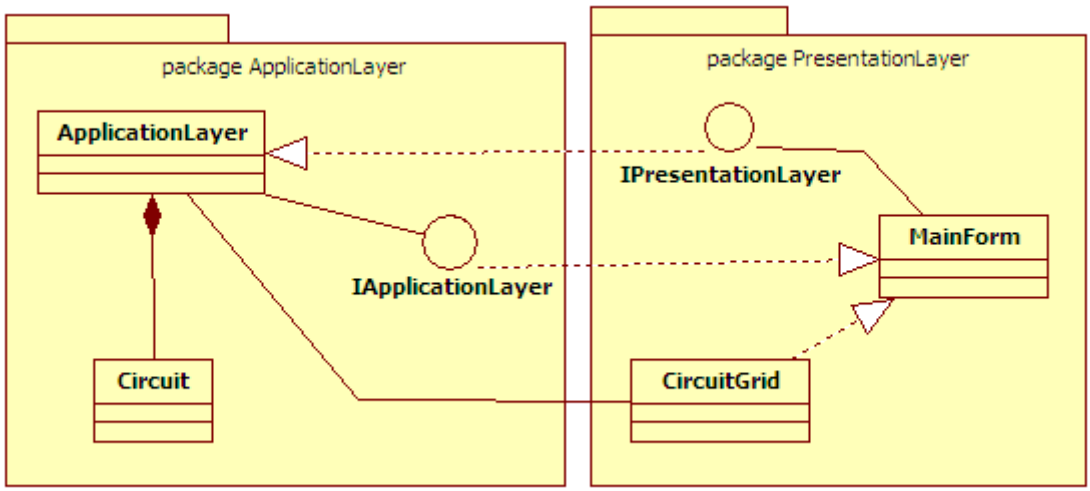
6 Vnitřní struktura aplikace

Tato kapitola obsahuje zjednodušený pohled určený pro nastínění fungování aplikace. Zájemci o konkrétní detaily si mohou zdrojový kód prohlédnout v Příloze nebo na internetu (http://kirchhoff.gfxs.cz).

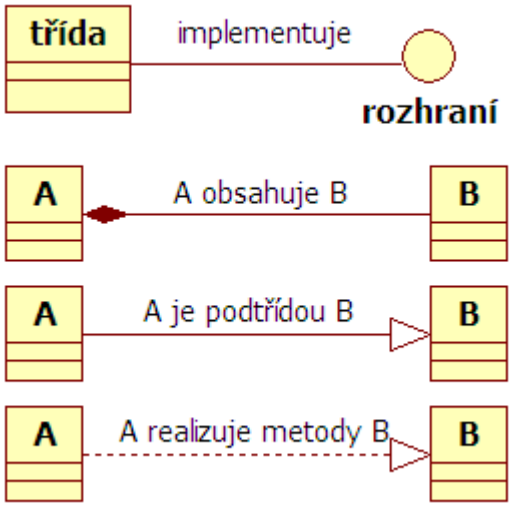
6.1 Základní rozvržení

Základní rozvržení (obr. 7) je zobrazeno formou UML třídního diagramu (stejnou formou jsou zobrazena i podrobnější schémata níže). Vysvětlivky k diagramům jsou na obr. 8.

Zdrojový kód je rozdělen do dvou balíků (*package*) *ApplicationLayer* a *PresentationLayer*. Balík *PresentationLayer* zpracovává kontakt s uživatelem (grafické prostředí, zpracovává vstupy, zobrazuje výstupy). Balík *ApplicationLayer* obstarává vnitřní operace aplikace (zpracování obvodu, generování a počítání rovnic). Hlavní třídy balíků komunikují přes rozhraní *IAplicationLayer* a *IPresentationLayer*. Data obvodů jsou uložena ve třídě *CircuitGrid* (formou mřížky) a *Circuit* (formou logického grafu).



Obrázek 7 – Základní rozvržení aplikace



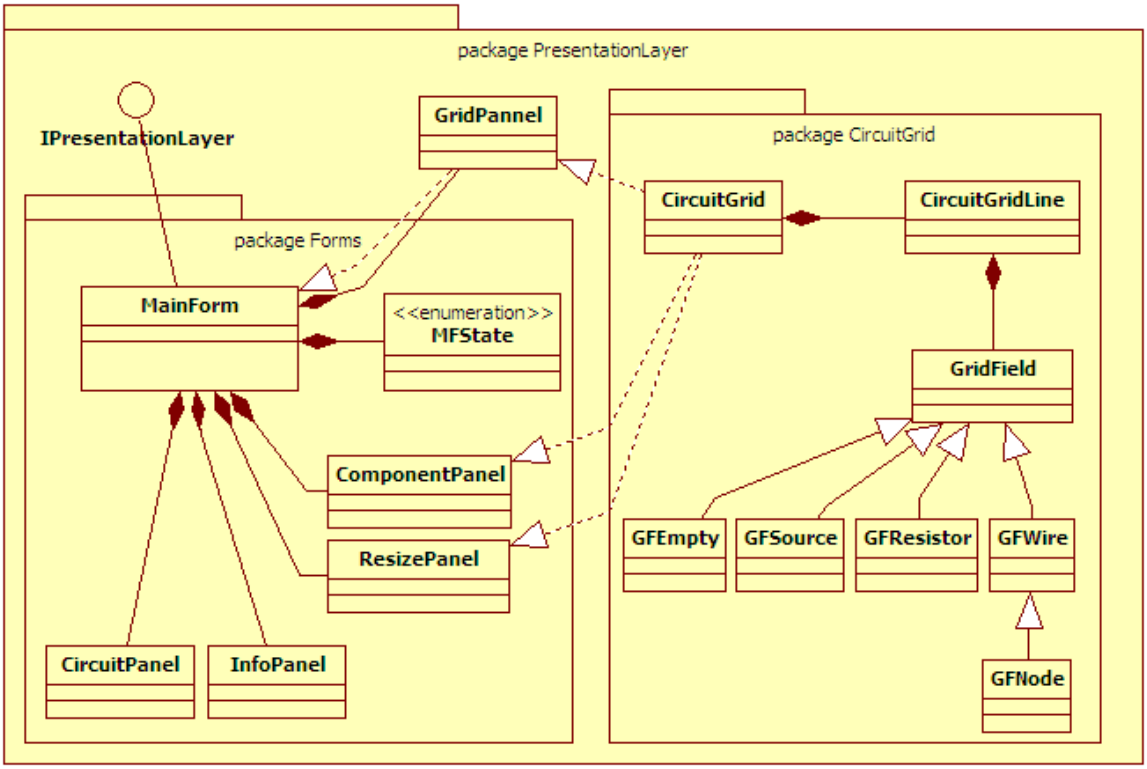
Obrázek 8 – Legenda k UML diagramům (Obrázky 7, 9, 10)

6.2 PresentationLayer

Balík *PresentationLayer* (obr. 9) je rozdělen na dva podbalíky: *Forms* a *CircuitGrid*.

Forms upravuje uživatelské rozhraní, jeho hlavní třídou je *MainForm* – hlavní formulář, do kterého jsou vloženy panely umožňující měnit obvod a informovat uživatele. Aktuální stav aplikace je určen zvolenou hodnotou z výčtu (*enumeration*) *MFState*. Uživatelské vstupy jsou zpracovávány třídami *ComponentPanel* (úprava komponent), *ResizePanel* (změna velikosti mřížky) a *GridPanel* (samotné kreslení částí obvodu).

CircuitGrid obsahuje všechny informace v mřížce. Hlavní třída téhož jména obsahuje do řádek uspořádané potomky třídy *GridField*, zastupující různé druhy jednotlivých políček mřížky (uzly, vodiče, prázdná pole, …).



Obrázek 9 – Struktura balíku PresentationLayer

6.3 ApplicationLayer

Balík *ApplicationLayer* (obr. 10) obsahuje stejnojmennou hlavní třídu, podbalík *Circuit* a několik dalších tříd.

Při zpracovávání vstupu předá třída *ApplicationLayer* data z mřížky třídě *ConvertToLogic*, která vytvoří logický ekvivalent obvodu – graf složený z uzlů (*Node*) a větví (*Branch*) obsahujících komponenty (*Component*).

Potom jsou metodou úplného stromu (podrobně popsána v [2]) vygenerovány smyčky (*Loop*). Existuje mnoho kombinací smyček, které řeší zadanou úlohu. Soustavy rovnic z nich vytvořené jsou ale rozdílně složité – mají sice vždy stejný počet rovnic, ale délka jejich zápisu může být zbytečně složitá. Proto aplikace najde větší počet systémů smyček (úměrný počtu větví) a porovnáním počtů členů rovnic vybere ty smyčky, které jich mají nejméně.

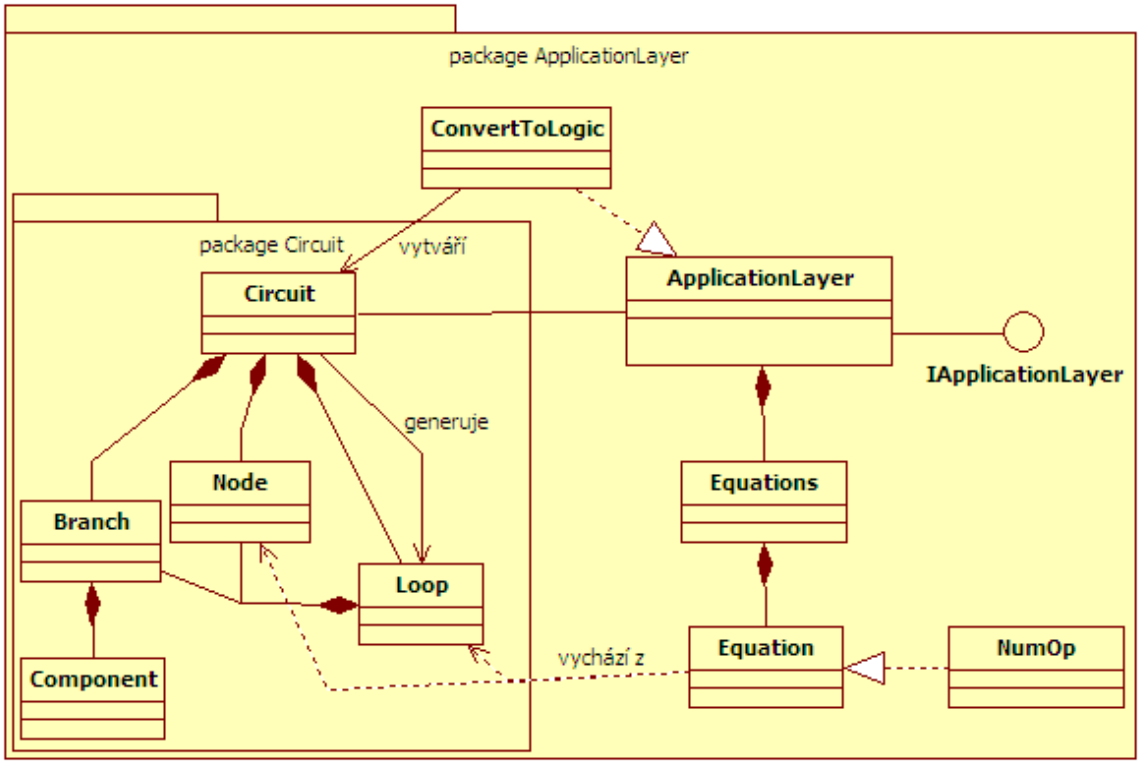
Když jsou smyčky hotovy, program vytvoří soustavu rovnic (*Equations*). Rovnice jsou uspořádány do matice a pomocí externího balíku Apache Commons Math[3] vypočítány.

Pokud celé zpracování obvodu proběhlo bez chyb, je uživatel proveden jednotlivými fázemi řešení úlohy. Pokud nastane chyba, znamená to, že navržený obvod má nějakou vadu.

[3] Více informací a zdrojové kódy knihovny na http://commons.apache.org/math/

Aplikace podle typu chyby, která se objeví během zpracování, určí vadu obvodu a oznámí uživateli pravděpodobnou vadu.

V balíku se nachází také třída *NumOp*, která zajišťuje operace s desetinnými čísly jako je zaokrouhlování na určitý počet desetinných míst či jejich formátování pro výstup.



Obrázek 10 – Struktura balíku ApplicationLayer

7 Závěr a diskuse

Úspěšně se mi podařilo vytvořit aplikaci Kirchhoff, sloužící k výuce základní látky Kirchhoffových zákonů. Krok za krokem provede uživatele základy teorie a řešením běžných úloh.

Aplikace je použitelná jak pro třídy, tak pro jednotlivce. Software je volně dostupný na internetu (http://kirchhoff.gfxs.cz).

Aplikace Kirchhoff je plně funkční. Jsou zde samozřejmě možnosti rozšíření – např. možnost ukládání a načítání vytvořených obvodů, nabídka vlastních úloh, implementace dalších typů úloh, tvorba anglické jazykové verze atd.

8 Seznam použité literatury

[1] ECKEL, Bruce. *Thinking in Java* [online]. 3rd edition. [s.l.] : Prentice-Hall, December 2002 [cit. 2010-06-03]. Dostupné z WWW: <http://www.mindview.net/Books/TIJ/>.

[2] JAREŠOVÁ, Miroslava. *Elektrické obvody (Stejnosměrný proud)* [online]. [s.l.] : [s.n.], 200? [cit. 2011-02-26]. Dostupné z WWW: <http://fo.cuni.cz/texty/elobvody.pdf>.

[3] Wikipedia contributors. Class diagram [online]. Wikipedia, The Free Encyclopedia; 2011 Feb 24, 12:26 UTC [cited 2011 Feb 26]. Dostupné z WWW: http://en.wikipedia.org/w/index.php?title=Class_diagram&oldid=415679311.

[4] Wikipedia contributors. Revision control [online]. Wikipedia, The Free Encyclopedia; 2011 Feb 21, 12:22 UTC [cited 2011 Feb 26]. Dostupné z WWW: http://en.wikipedia.org/w/index.php?title=Revision_control&oldid=415119797.

[5] Wikipedia contributors. Apache Subversion [online]. Wikipedia, The Free Encyclopedia; 2011 Feb 3, 14:34 UTC [cited 2011 Feb 26]. Dostupné z WWW: http://en.wikipedia.org/w/index.php?title=Apache_Subversion&oldid=411784362.

[6] Wikipedia contributors. TortoiseSVN [online]. Wikipedia, The Free Encyclopedia; 2010 Dec 19, 22:34 UTC [cited 2011 Feb 26]. Dostupné z WWW: http://en.wikipedia.org/w/index.php?title=TortoiseSVN&oldid=403246633.

Příloha – zdrojový kód aplikace Kirchhoff

Tato příloha obsahuje okomentovaný zdrojový kód aplikace vytvořený autorem (bez kódu externí knihovny). Jednotlivé soubory jsou odděleny nadpisem **Soubor** a relativní adresou souboru. Struktura složek odpovídá struktuře balíků popsané v kapitole č. 6.

Zdrojový kód je také ke stažení na stránkách http://kirchhoff.gfxs.cz.

Obsah přílohy

Soubor: ./src/ApplicationLayer/ApplicationLayer.java

```
package ApplicationLayer;

import ApplicationLayer.Circuit.Circuit;
import PresentationLayer.*;
import PresentationLayer.Forms.MainForm;
import PresentationLayer.Grid.CircuitGrid;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import
org.apache.commons.math.linear.SingularMatrixException
;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class ApplicationLayer implements
IApplicationLayer {

    public static ApplicationLayer self = new
ApplicationLayer();
    private IPresentationLayer pl;
    private CircuitGrid cg;
    private Circuit c;
    private Equations eqs;

    private ApplicationLayer() {
    }

    /**
     * spuštění a inicializace
     */
    public void run() {
        cg = CircuitGrid.self;
        eqs = Equations.self;
        pl = new MainForm();
    }

    /**
     * zpracuje obvod, vytvoří a vypočítá rovnice (aby
se zjistilo, zda je správně navržen)
     * nechá zobrazit prvky obvodu
     */
    public void processCG() {
        cg.setMarks();
        c = ConvertToLogic.self.convert();
        try {
            c.markBranches();
            c.generateLoopSystems();
            eqs.generateEquations(c);
            eqs.count(c);
            pl.showBranches(c);
        } catch (IndependentCircuitsException e) {
            pl.showErrorMessage("Obvod se
pravděpodobně skládá z více nezávislých obvodů a nemá
smysl jej počítat jako celek.");
        } catch (ShortCircuitException e) {
            pl.showErrorMessage("Obvod je zkratovaný
nebo obsahuje paralelní prázdné větve!\nObvod obsahuje
smyčku s nulovým odporem a nemá smysl jej
počítat.\nOdstraňte přebytečné prázdné větve nebo
doplňte chybějící odpory.");
        } catch (EmptyLoopException e) {
            pl.showErrorMessage("Obvod se
pravděpodobně skládá z více nezávislých obvodů a nemá
smysl jej počítat jako celek.");
        } catch (SingularMatrixException e) {
            pl.showErrorMessage("Obvod se
pravděpodobně skládá z více nezávislých obvodů a nemá
smysl jej počítat jako celek.");
        }
    }

    /**
     * nechá zobrazit rovnice
     */
    public void generateEquations() {
        pl.showEquations(eqs.equations);
    }

    /**
     * nechá zobrazit řešení rovnic
     */
    public void solveEquations() {
        eqs.saveSolution(c);
        pl.showSolution(eqs.getSolution(),
c.needInterpretSolution());
    }

    /**
     *  interpretuje výsledky (záporné a nulové
hodnoty proudu)
     */
    public void interpretSolution() {
        c.interpretSolution();
        pl.interpretSolution();
    }

    public static String loadFileToString(String path)
{
        InputStream in = null;
        String lines = "";
        try {
            in =
MainForm.class.getResourceAsStream(path);
            BufferedReader reader = new
BufferedReader(new InputStreamReader(in));
            String line = null;
            while ((line = reader.readLine()) != null)
{
                lines += line;
                //System.out.println(line);
            }
        } catch (IOException ex) {
            System.err.println(ex);
        } finally {
            if (in != null) {
                try {
                    in.close();
                } catch (IOException ex) {
                }
            }
        }
        return lines;
    }
}
```

Soubor: ./src/ApplicationLayer/Circuit/Branch.java

```
package ApplicationLayer.Circuit;

import ApplicationLayer.NumOp;
import ApplicationLayer.Position;
import ApplicationLayer.Direction;
import PresentationLayer.Grid.CircuitGrid;
import PresentationLayer.Grid.GFNode;
import java.util.ArrayList;
import java.util.Iterator;
```

```
/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Branch implements Highlightable {

    public ArrayList<Node> nodes = new
ArrayList<Node>(2);
    public ArrayList<Direction> directions = new
ArrayList<Direction>(2);//směry, kterými větev začíná
a končí (které části uzlů patří k větvi)
    public ArrayList<Component> components = new
ArrayList<Component>(8);
    private ArrayList<Position> positions = new
ArrayList<Position>();
    public int current = 1;//směr proudu: 1 = od
nodes.get(0) do nodes.get(1)
    private double I;
    private int mark;

    public Branch(Node nodeA, Direction dirA, Node
nodeB, Direction dirB, ArrayList<Component>
components, ArrayList<Position> positions) {
        nodes.add(nodeA);
        nodes.add(nodeB);
        directions.add(dirA);
        directions.add(dirB);
        this.components = components;
        this.positions = positions;
    }

    public boolean hasNode(Node node) {
        return (nodes.contains(node));
    }

    /**
     * @param node jeden uzel
     * @return ten druhý uzel/null, pokud ho
neobsahuje
     */
    public Node otherNode(Node node) {
        if (hasNode(node)) {
            return (node == nodes.get(0)) ?
nodes.get(1) : nodes.get(0);
        } else {
            return null;
        }
    }

    /**
     * @return směr větve od daného uzlu/null, pokud
uzel neobsahuje
     */
    public Direction getDirection(Node node) {
        if (nodes.get(0).equals(node)) {
            return directions.get(0);
        }
        if (nodes.get(1).equals(node)) {
            return directions.get(1);
        }
        return null;
    }

    /**
     * @return označení smyčky - název do listu
     */
    @Override
    public String toString() {
        return ("Větev " + mark/* + "\n   " +
nodes.get(0).toString() + " Dir: " + directions.get(0)
+ "\n Comp: " + components.size() + "\n   " +
nodes.get(1).toString() + " Dir: " + directions.get(1)
+ "\n---\n"*/);
    }

    /**
     * @return pozice polí větve včetně uzlů
     */
    public ArrayList<Position> getPositions() {
        ArrayList<Position> poss = new
ArrayList<Position>();
        poss.add(nodes.get(0).pos);
        poss.addAll(positions);
        poss.add(nodes.get(1).pos);
        return poss;
    }

    /**
     * @return pozice polí v pořadí od daného uzle
(včetně daného uzle)
     */
    public ArrayList<Position> getPositions(Node
fromNode) {
        ArrayList<Position> poss = new
ArrayList<Position>();
        if (nodes.get(0).equals(fromNode)) {
            poss.add(nodes.get(0).pos);
            poss.addAll(positions);
        } else {
            poss.add(nodes.get(1).pos);
            for (int i = positions.size() - 1; i >= 0;
i--) {
                poss.add(positions.get(i));
            }
        }
        return poss;
    }

    /**
     * @return celkový odpor větve (vždy kladný)
     */
    public double getResistance() {
        double resistance = 0;
        Iterator<Component> it =
components.iterator();
        Component c;
        while (it.hasNext()) {
            c = it.next();
            if (!c.isSource) {
                resistance += c.resistance;
            }
        }
        return NumOp.roundDouble(resistance);
    }

    /**
     * @return součet napětí zdrojů ve větvi větvi
(kladný, pokud + směřuje k nodes.get(1))
     */
    public double getVoltage() {
        double voltage = 0;
        Iterator<Component> it =
components.iterator();
        Component c;
        while (it.hasNext()) {
            c = it.next();
            if (c.isSource) {
                voltage += (c.plusNode ==
nodes.get(1)) ? c.voltage : -c.voltage;
            }
        }
        return NumOp.roundDouble(voltage);
    }

    public Node getFirstNode() {
        return (current != -1) ? nodes.get(0) :
nodes.get(1);
    }
```

```
    /**
     * @return the I
     */
    public double getI() {
        return I;
    }

    /**
     * @param I the I to set
     */
    public void setI(double I) {
        this.I = I;
    }

    /**
     * @return potřebuje změnit směr v nákresu?
     */
    public boolean needInterpret() {
        return (I <= 0);
    }

    /**
     * změní směry proudu v nákresu, aby odpovídaly
kladé hodnotě proudu
     * odstraní směr proudu v nákresu, pokud proud
větví neteče
     */
    public void interpretSolution() {
        if (I < 0) {
            current = -current;
            this.I *= -1;
            for (int i = 0; i <= 1; i++) {
                Node node = nodes.get(i);
                ((GFNode)
CircuitGrid.self.get(node.pos)).invertCurrent(directio
ns.get(i));
            }
        }
        if (I == 0) {
            current = 0;
            for (int i = 0; i <= 1; i++) {
                Node node = nodes.get(i);
                ((GFNode)
CircuitGrid.self.get(node.pos)).setCurrent(directions.
get(i), 0);
            }
        }
    }

    /**
     * @return the mark
     */
    public int getMark() {
        return mark;
    }

    /**
     * @param mark the mark to set
     */
    public void setMark(int mark) {
        this.mark = mark;
        nodes.get(0).node.setBranch(directions.get(0),
mark);
        nodes.get(1).node.setBranch(directions.get(1),
mark);
    }
}
```

Soubor: ./src/ApplicationLayer/Circuit/Circuit.java

```
package ApplicationLayer.Circuit;
import ApplicationLayer.ApplicationLayer;
import ApplicationLayer.EmptyLoopException;
import ApplicationLayer.Equations;
import ApplicationLayer.IndependentCircuitsException;
import ApplicationLayer.NumOp;
import ApplicationLayer.Position;
import ApplicationLayer.ShortCircuitException;
import ApplicationLayer.Direction;
import PresentationLayer.Forms.MFState;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Circuit {

    public ArrayList<Node> nodes = new
ArrayList<Node>(20);
    public ArrayList<Branch> branches = new
ArrayList<Branch>(20);
    public ArrayList<Branch> tree = new
ArrayList<Branch>(20);
    public ArrayList<Branch> outoftree = new
ArrayList<Branch>();
    public ArrayList<Loop> loops = new
ArrayList<Loop>();
    private ArrayList<ArrayList<Branch>> trees = new
ArrayList<ArrayList<Branch>>();
    private ArrayList<ArrayList<Branch>> outOfTrees =
new ArrayList<ArrayList<Branch>>();
    private ArrayList<ArrayList<Loop>> loopSystems =
new ArrayList<ArrayList<Loop>>();

    /**
     * vytvoří prázdný obvod
     */
    public Circuit() {
    }

    public Circuit(ArrayList<Node> nodes,
ArrayList<Branch> branches, ArrayList<Branch> tree,
ArrayList<Branch> outoftree, ArrayList<Loop> loops) {
        this.nodes = nodes;
        this.branches = branches;
        this.tree = tree;
        this.outoftree = outoftree;
        this.loops = loops;
    }

    public boolean hasNode(Position pos) {
        Iterator<Node> it = nodes.iterator();
        while (it.hasNext()) {
            if (it.next().pos.is(pos)) {
                return true;
            }
        }
        return false;
    }

    public Node getNode(Position pos) {
        Iterator<Node> it = nodes.iterator();
        Node node;
        while (it.hasNext()) {
            node = it.next();
            if (node.pos.is(pos)) {
                return node;
            }
        }
        return null;
    }
```

```
    public boolean hasBranch(Node node, Direction dir)
{
        Iterator<Branch> it = branches.iterator();
        Branch br;
        while (it.hasNext()) {
            br = it.next();
            if (br.nodes.get(0).pos.is(node.pos) &
br.directions.get(0).ordinal() == dir.ordinal()) {
                return true;
            }
            if (br.nodes.get(1).pos.is(node.pos) &
br.directions.get(1).ordinal() == dir.ordinal()) {
                return true;
            }
        }
        return false;
    }

    /**
     * @param node výchozí uzel
     * @param unmarked true/false: pouze neoznačené
větve/všechny větve
     * @return větve vycházející z tohoto uzlu
     */
    public ArrayList<Branch> branches(Node node,
boolean unmarked) {
        ArrayList<Branch> nextBr = new
ArrayList<Branch>();
        Iterator<Branch> itBr = branches.iterator();
        while (itBr.hasNext()) {
            Branch br = itBr.next();
            if (br.hasNode(node)) {
                if (!unmarked | br.getMark() == 0) {
                    nextBr.add(br);
                }
            }
        }
        return nextBr;
    }

    /**
     * @return počet potřebných rovnic vyplývajících z
1. Kirchhoffova zákona
     */
    public int firsLawEquationsNeeded() {
        return (nodes.size() - 1);
    }

    /**
     * @return počet potřebných rovnic vyplývajících z
2. Kirchhoffova zákona
     */
    public int secondLawEquationsNeeded() {
        return (branches.size() - nodes.size() + 1);
    }

//-----------------------------------------------------
-------------------------
    /**
     * nalezne různé úplné stromy el. sítě
     */
    private void findTrees() {
        trees.clear();
        Node node;
        //System.out.println("findTrees");
        for (int n = 0; n < nodes.size(); n++) {
            node = nodes.get(n);
            //System.out.println("\n node" + n);
            for (int b = 0; b < branches(node,
false).size(); b++) {
                //System.out.println("  " + b);
                tree.clear();
                outoftree = (ArrayList<Branch>)
branches.clone();
                firstTreeBranch(node, b);
                trees.add((ArrayList<Branch>)
tree.clone());
                outOfTrees.add((ArrayList<Branch>)
outoftree.clone());
            }
        }
        //System.out.println("Trees found:" + trees);
    }

    /**
     * začne vyhledávání stromu určiou větví (potřeba
pro nalezení různých stromů)
     */
    private void firstTreeBranch(Node node, int index)
{
        ArrayList<Branch> posible = branches(node,
false);
        Branch br = posible.get(index);
        //System.out.println("   First branch: " +
br);
        tree.add(br);
        outoftree.remove(br);
        //System.out.println("FindTreeBranch" +
toString());
        findTreeBranch(br.otherNode(node));
    }

    /**
     * postupně přidává větve vedoucí k uzlům dosud
neobsaženým ve stromu, rekurze pro nově připojené uzly
     * @param node výchozí uzel
     */
    private void findTreeBranch(Node node) {
        ArrayList<Branch> posible = branches(node,
false);
        Iterator<Branch> it = posible.iterator();
        Branch br;
        while (it.hasNext()) {
            br = it.next();
            if (!treeHasNode(br.otherNode(node))) {
                tree.add(br);
                outoftree.remove(br);
                //System.out.println("FindTreeBranch"
+ toString());
                findTreeBranch(
                        br.otherNode(node));
            }
        }
    }

    /**
     * zjistí, zda-li je dotazovaný uzel obsažen ve
stromu
     */
    private boolean treeHasNode(Node node) {
        Iterator<Branch> itBr = tree.iterator();
        Branch br;
        while (itBr.hasNext()) {
            br = itBr.next();
            if (br.hasNode(node)) {
                return true;
            }
        }
        return false;
    }

    /**
     * vyhledá stromy sítě, vygeneruje systémy smyček
(pro každý strom) a poté vybere 1 systém
     */
```

```
    public void generateLoopSystems() throws
ShortCircuitException, EmptyLoopException {
        findTrees();
        loopSystems.clear();
        for (int i = 0; i < trees.size(); i++) {
            //System.out.println("generating loop
system " + i);
            loops.clear();
            Iterator<Branch> it =
outOfTrees.get(i).iterator();
            if (outOfTrees.get(i).size() !=
secondLawEquationsNeeded()) {
                //System.out.println("Number of
outoftree branches doesn't match!");
            }
            while (it.hasNext()) {
                loops.add(new Loop(nodes,
trees.get(i), it.next()));
                //System.out.println("New Loop"/* +
toString()*/);
            }
            //System.out.println(loops);
            loopSystems.add((ArrayList<Loop>)
loops.clone());
        }
        selectLoopSystem();
    }

    /**
     * vybere systém smyček s nejmenší složitostí
soustavy rovnic
     */
    private void selectLoopSystem() throws
ShortCircuitException {
        //System.out.println("Selecting loop System");
        int lastSize = Equations.getPredictedSize(new
Circuit(nodes, branches, trees.get(0),
outOfTrees.get(0), loopSystems.get(0)));
        int thisSize;
        tree = trees.get(0);
        outoftree = outOfTrees.get(0);
        loops = loopSystems.get(0);
        //System.out.println("size 0 : " + lastSize);
        for (int i = 1; i < trees.size(); i++) {
            thisSize = Equations.getPredictedSize(new
Circuit(nodes, branches, trees.get(i),
outOfTrees.get(i), loopSystems.get(i)));
            //System.out.println("size " + i + " : " +
thisSize);
            if (thisSize < lastSize) {
                tree = trees.get(i);
                outoftree = outOfTrees.get(i);
                loops = loopSystems.get(i);
                lastSize = thisSize;
            }
        }
        //System.out.println("Selected size: " +
lastSize);
    }

    /**
     * označí větve čísly
     * postupuje od uzlu A, pro každý uzel označí jeho
zatím neoznačené větve po směru hodinových ručiček
     */
    public void markBranches() throws
IndependentCircuitsException {
        orderNodes();
        Node node;
        Branch selected;
        Branch br;
        Iterator<Branch> it;
        Iterator<Node> itN = nodes.iterator();
        node = itN.next();
        //System.out.println("markBranches");
        //System.out.println(" " + node.toString());
        for (int i = 0; i < branches.size(); i++) {
            it = branches(node, true).iterator();
            //System.out.println(" " + node.toString()
+ " " + i);
            //System.out.println("  " + branches(node,
true));
            try {
                selected = it.next();
            } catch (NoSuchElementException e) {
                throw new
IndependentCircuitsException();
            }
            //System.out.println("  sel:" +
selected.toString() + " has next " + it.hasNext());
            if (!it.hasNext()) {
                if (itN.hasNext()) {
                    node = itN.next();
                }
            }
            while (it.hasNext()) {
                br = it.next();
                if (br.getDirection(node).ordinal() <
selected.getDirection(node).ordinal()) {
                    selected = br;
                }
            }
            selected.setMark(i + 1);
        }
        orderBranches();
    }

    /**
     * seřadí větve v arraylistu podle označení
     */
    private void orderBranches() {
        ArrayList<Branch> bs = (ArrayList<Branch>)
branches.clone();
        branches.clear();
        for (int i = 1; i <= bs.size(); i++) {
            branches.add(getBranch(i, bs));
        }
    }

    /**
     * seřadí uzly v arraylistu podle označení
     */
    private void orderNodes() {
        ArrayList<Node> ns = (ArrayList<Node>)
nodes.clone();
        nodes.clear();
        for (char c = 'A'; c <= 'A' + ns.size() - 1;
c++) {
            nodes.add(getNode(c, ns));
        }
    }

    /**
     * @return větev s daným označením z daného
arraylistu/null pokud tam taková není
     */
    private Branch getBranch(int hasMark,
ArrayList<Branch> bs) {
        Iterator<Branch> it = bs.iterator();
        Branch b;
        while (it.hasNext()) {
            b = it.next();
            if (b.getMark() == hasMark) {
                return b;
            }
        }
        System.out.println("Error in
Circuit:getBranch(int)");
```

```
        return null;
    }

    /**
     * @return uzel s daným označením z daného
arraylistu/null pokud tam takový není
     */
    private Node getNode(char hasMark, ArrayList<Node>
ns) {
        Iterator<Node> it = ns.iterator();
        Node n;
        while (it.hasNext()) {
            n = it.next();
            if (n.node.getCharMark() == hasMark) {
                return n;
            }
        }
        System.out.println("Error in
Circuit:getBranch(int)");
        return null;
    }

//-------------------------------------------------
-------------------------
    /**
     * pro potřeby výpisů
     */
    @Override
    public String toString() {
        String s = "\nCircuit";
        Iterator<Node> itN = nodes.iterator();
        while (itN.hasNext()) {
            s = s.concat(itN.next().toString());
        }
        Iterator<Branch> itB = branches.iterator();
        while (itB.hasNext()) {
            s = s.concat(itB.next().toString());
        }
        s = s.concat("\nTree");
        itB = tree.iterator();
        while (itB.hasNext()) {
            s = s.concat(itB.next().toString());
        }
        s = s.concat("\nOut of Tree");
        itB = outoftree.iterator();
        while (itB.hasNext()) {
            s = s.concat(itB.next().toString());
        }
        Iterator<Loop> itL = loops.iterator();
        while (itL.hasNext()) {
            s = s.concat(itL.next().toString());
        }
        return (s + "\n--------");
    }

    /**
     * @return html string popis
prvku(uzel,větev,smyčka zatím prázný) obvodu
     */
    public String getPartDescription(MFState state,
Object part) {
        String s = "";
        s =
ApplicationLayer.loadFileToString("html/header.html");
        if (part.getClass() == Node.class) {
            Node n = (Node) part;
            s = s.concat("<h3>Uzel " +
n.node.getMark() + "</h3>");
            if (state == MFState.FORMING_EQUATIONS |
state == MFState.EQUATIONS) {
                int branch;
                boolean dir;
                s += "<p>Podle zvolených směrů
proudu:</p>";
                String sDo = "";
                String sZ = "";
                for (int i = 0; i < 4; i++) {
                    branch =
n.node.getCurrent(Direction.values()[i]);
                    if (branch > 0) {
                        sDo += "<i>I</i><sub>" +
n.node.branches[i] + "</sub><br>";
                    }
                    if (branch < 0) {
                        sZ += "<i>I</i><sub>" +
n.node.branches[i] + "</sub><br>";
                    }
                }
                s += "<p>Do uzlu vchází proud/y</p><p
class=\"tab\">" + sDo + "</p><p>Z uzlu odchází
proud/y</p><p class=\"tab\"> " + sZ + "</p>";
            }
            if (state.ordinal()>2) {
                int branch;
                boolean dir;
                String sDo = "";
                String sZ = "";
                for (int i = 0; i < 4; i++) {
                    branch =
n.node.getCurrent(Direction.values()[i]);
                    if (branch > 0) {
                        sDo += "<i>I</i><sub>" +
n.node.branches[i] + "</sub> = " +
NumOp.writeDouble(branches.get(n.node.branches[i] -
1).getI()) + " A<br>";
                    }
                    if (branch < 0) {
                        sZ += "<i>I</i><sub>" +
n.node.branches[i] + "</sub> = " +
NumOp.writeDouble(branches.get(n.node.branches[i] -
1).getI()) + " A<br>";
                    }
                }
                s += "<p>Do uzlu vchází proud/y</p><p
class=\"tab\">" + sDo + "</p><p>Z uzlu odchází
proud/y</p><p class=\"tab\"> " + sZ + "</p>";
            }
        }
        if (part.getClass() == Branch.class) {
            Branch b = (Branch) part;
            Iterator<Component> it =
b.components.iterator();
            Component comp;
            String comps = "<h4>Komponenty</h4>";
            s += "<h3>Větev " + b.getMark() + "</h3>";
            if (state == MFState.FORMING_EQUATIONS |
state == MFState.EQUATIONS) {
                s += "<p>Větví prochází proud
<i>I</i><sub>" + b.getMark() + "</sub></p>";
                while (it.hasNext()) {
                    comp = it.next();
                    if (comp.isSource) {
                        comps +=
"<p><strong>Zdroj</strong> <i>U</i> = " +
NumOp.writeDouble(comp.voltage) + " V</p>";
                    } else {
                        comps +=
"<p><strong>Rezistor</strong> <i>R</i> = " +
NumOp.writeDouble(comp.resistance) + " Ω</p>";
                    }
                }
            }
            if (state.ordinal()>2) {
                s += "<p>Větví prochází proud
<i>I</i><sub>" + b.getMark() + "</sub> = " +
NumOp.writeDouble(b.getI()) + " A</p>";
                while (it.hasNext()) {
                    comp = it.next();
```

```
                    if (comp.isSource) {
                        comps +=
"<p><strong>Zdroj</strong> <i>U</i> = " +
NumOp.writeDouble(comp.voltage) + " V</p>";
                    } else {
                        comps +=
"<p><strong>Rezistor</strong> <i>R</i> = " +
NumOp.writeDouble(comp.resistance) + " Ω;
<i>U</i> = " +
NumOp.writeDouble(Math.abs(NumOp.roundDouble(comp.resi
stance * b.getI()))) + " V</p>";
                    }
                }
            }
            s += "<p>Celkový odpor větve je
<i>R</i> = " +
NumOp.writeDouble(b.getResistance()) + " Ω</p>";
            s += comps;
        }
        if (part.getClass() == Loop.class) {
            Loop l = (Loop) part;
        }
        return s;
    }

    /**
     * @return potřebuje některá část změnit směr v
nákresu?
     */
    public boolean needInterpretSolution() {
        Iterator<Branch> it = branches.iterator();
        boolean need = false;
        while (it.hasNext()) {
            if (it.next().needInterpret())need = true;
        }
        return need;
    }

    /**
     * změní směry proudů v nákresu, aby odpovídaly
kladým hodnotám proudů
     * odstraní směry proudů v nákresu, pokud proudy
větví netečou
     */
    public void interpretSolution() {
        Iterator<Branch> it = branches.iterator();
        while (it.hasNext()) {
            it.next().interpretSolution();
        }
    }
}
```

Soubor: ./src/ApplicationLayer/Circuit/Component.java

```
package ApplicationLayer.Circuit;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Component {

    public boolean isSource;//zdroj/rezistor
    public double resistance;
    public double voltage;
    public Node plusNode;//ke kterému uzlu ukazuje +
pól zdroje

    /**
     * vytvoř rezistor
     * @param resistance
     */
    public Component(double resistance) {
        isSource = false;
        this.resistance = resistance;
    }

    /**
     * vytvoř zdroj
     * @param voltage
     * @param plusNode ke kterému uzlu ukazuje + pól
zdroje
     */
    public Component(double voltage, Node plusNode) {
        isSource = true;
        this.voltage = voltage;
        this.plusNode = plusNode;
    }
}
```

Soubor:
./src/ApplicationLayer/Circuit/Highlightable.java

```
package ApplicationLayer.Circuit;

import ApplicationLayer.Position;
import java.util.ArrayList;

/**
 * @author Jakub Hrncir (2010)
 */
public interface Highlightable {
    /**
     * pozice v mřížce pro zvýraznění
     * @return
     */
    public ArrayList<Position> getPositions();
}
```

Soubor: ./src/ApplicationLayer/Circuit/Loop.java

```
package ApplicationLayer.Circuit;

import ApplicationLayer.Position;
import ApplicationLayer.EmptyLoopException;
import java.util.ArrayList;
import java.util.Iterator;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Loop implements Highlightable {

    public ArrayList<Branch> branches = new
ArrayList<Branch>();
    public ArrayList<Node> nodes = new
ArrayList<Node>();
    private ArrayList<Position> positions = new
ArrayList<Position>();

    public Loop(ArrayList<Node> nodes,
ArrayList<Branch> tree, Branch branch) throws
EmptyLoopException {
        this.nodes = (ArrayList<Node>) nodes.clone();
        branches = (ArrayList<Branch>) tree.clone();
        branches.add(branch);
        while (removeExcessiveBranches()) {
        }
        if (branches.isEmpty()) {
            throw new EmptyLoopException();
        }
        order();
        setPositions();
    }

    /**
```

```
     * odstraní přebývající větev smyčky (while
cyklus)
     * @return byla odstraněna větev?
     */
    private boolean removeExcessiveBranches() {
        Iterator<Node> it = nodes.iterator();
        Node node;
        boolean removed = false;
        while (!removed & it.hasNext()) {
            node = it.next();
            if (branches(node).size() < 2) {
                removeBranch(node);
                removed = true;
            }
        }
        //System.out.println("Remove excessive
branches: " + removed + " Branches: " +
branches.size());
        return removed;
    }

    /**
     * odstraní větev vedocí z výchozího uzlu (pouze
jednu)
     */
    private void removeBranch(Node node) {
        Iterator<Branch> it = branches.iterator();
        Branch br;
        boolean removed = false;
        while (!removed && it.hasNext()) {
            br = it.next();
            if (br.hasNode(node)) {
                branches.remove(br);
                removed = true;
            }
        }
        nodes.remove(node);
    }

    /**
     * @param node výchozí uzel
     * @return větve vycházející z tohoto uzlu
     */
    private ArrayList<Branch> branches(Node node) {
        ArrayList<Branch> brs = new
ArrayList<Branch>();
        Iterator<Branch> itBr = branches.iterator();
        while (itBr.hasNext()) {
            Branch br = itBr.next();
            if (br.hasNode(node)) {
                brs.add(br);
            }
        }
        return brs;
    }

    /**
     * @return označení smyčky - název do listu
     */
    @Override
    public String toString() {
        String s = "Smyčka ";
        Iterator<Node> itN = nodes.iterator();
        Iterator<Branch> itB = branches.iterator();
        while (itN.hasNext()) {
            s = s.concat(itN.next().node.getMark() +
",");
            s =
s.concat(Integer.toString(itB.next().getMark()) +
",");
        }
        return (s.substring(0, s.length() - 1));
    }

    /**
     * uloží do listu positions pozice potřebné k
zvýraznění smyčky
     */
    private void setPositions() {
        positions.clear();
        Iterator<Node> itN = nodes.iterator();
        Iterator<Branch> itB = branches.iterator();
        while (itN.hasNext()) {

positions.addAll(itB.next().getPositions(itN.next()));
        }
    }

    /**
     * seřadí uzly a větve v listech aby odpovídaly
pořadí při procházení smyčky
     */
    private void order() {
        ArrayList<Branch> list = (ArrayList<Branch>)
branches.clone();
        Iterator<Branch> it;
        Branch br;
        branches.clear();
        branches.add(list.remove(0));
        nodes.clear();
        nodes.add(branches.get(0).nodes.get(0));
        int end = list.size();
        Node node = branches.get(0).nodes.get(1);
        for (int i = 1; i <= end; i++) {
            nodes.add(node);
            it = list.iterator();
            while (i == branches.size() &&
it.hasNext()) {
                br = it.next();
                if (br.hasNode(node)) {
                    branches.add(br);
                    list.remove(br);
                    node = br.otherNode(node);
                }
            }
        }
    }

    public ArrayList<Position> getPositions() {
        return positions;
    }
}
```

Soubor: ./src/ApplicationLayer/Circuit/Node.java

```
package ApplicationLayer.Circuit;

import ApplicationLayer.Position;
import PresentationLayer.Grid.GFNode;
import java.util.ArrayList;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Node implements Highlightable {

    public Position pos;
    public GFNode node;

    public Node(Position pos, GFNode node) {
        this.pos = new Position(pos.row, pos.col);
        this.node = node;
    }

    /**
     * @return označení smyčky - název do listu
     */
```

```
    @Override
    public String toString() {
        return ("Uzel " + node.getMark()/* + " [" +
pos.row + "," + pos.col + "]"*/);
    }

    public ArrayList<Position> getPositions() {
        ArrayList<Position> positions = new
ArrayList<Position>();
        positions.add(pos);
        return positions;
    }
}
```

Soubor: ./src/ApplicationLayer/ConvertToLogic.java

```
package ApplicationLayer;

import ApplicationLayer.Circuit.Branch;
import ApplicationLayer.Circuit.Circuit;
import ApplicationLayer.Circuit.Component;
import ApplicationLayer.Circuit.Node;
import PresentationLayer.Grid.CircuitGrid;
import PresentationLayer.Grid.GFNode;
import PresentationLayer.Grid.GFResistor;
import PresentationLayer.Grid.GFSource;
import PresentationLayer.Grid.GFWire;
import PresentationLayer.Grid.GridField;
import java.util.ArrayList;
import java.util.Iterator;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class ConvertToLogic {

    public static ConvertToLogic self = new
ConvertToLogic();
    private CircuitGrid cg = CircuitGrid.self;
    private Circuit c;

    /**
     * převede oblod z mřížky do logické struktury
     */
    public Circuit convert() {
        c = new Circuit();
        Position pos = findNode();
        c.nodes.add(new Node(new Position(pos),
(GFNode) cg.get(pos)));
        findBranches(c.nodes.get(0));
        Iterator<Node> it = c.nodes.iterator();
        while (it.hasNext()) {
            ((GFNode)
cg.get(it.next().pos)).setKnowCurrents(true);
        }
        return c;
    }

    /**
     * vybere všechny dosud neznámé větve vycházející
ze vstupního uzlu a načte je
     */
    private void findBranches(Node node) {
        if (node == null) {
            return;
        }
        ArrayList<Direction> dirs =
getNode(node).getDirs();
        Iterator<Direction> it = dirs.iterator();
        Direction dir;
        while (it.hasNext()) {
            dir = it.next();
            if (!c.hasBranch(node, dir)) {
                findBranches(findBranch(node, dir));
            }
        }

    }

    /**
     * @return uzel, který je první na řadě při čtení
mřížky po řádcích
     */
    private Position findNode() {
        for (int row = 0; row < cg.getRowCount();
row++) {
            for (int col = 0; col < cg.getColCount();
col++) {
                if (cg.get(row, col).getClass() ==
GFNode.class) {
                    return new Position(row, col);
                }
            }
        }
        return new Position(0, 0);
    }

    /**
     * projde políčko po políčku větev určenou
vstupním uzlem a směrem
     * načte novou větev a zvolí směr proudu
     * načte nový uzel (pokud najde)
     * @return nový uzel, pokud uzel na konci hledané
větve není nový, vrací null
     */
    private Node findBranch(Node startNode, Direction
startDir) {
        Position pointer = new
Position(startNode.pos);
        ArrayList<Position> positions = new
ArrayList<Position>();//seznam políček větve
        ArrayList<Component> components = new
ArrayList<Component>();//seznam komponent větve
        pointer = move(startDir, pointer);
        Direction lastDir = startDir;
        while (cg.get(pointer).getClass() !=
GFNode.class) {
            positions.add(new Position(pointer));
            if (cg.get(pointer).getClass() ==
GFResistor.class) {
                components.add(new
Component(((GFResistor)
cg.get(pointer)).getResistance()));
            }
            if (cg.get(pointer).getClass() ==
GFSource.class) {
                components.add(new
Component(((GFSource) cg.get(pointer)).getVoltage(),
((((GFSource) cg.get(pointer)).getPolarity() &
(lastDir.ordinal() == 1 | lastDir.ordinal() == 2)) |
(!((GFSource) cg.get(pointer)).getPolarity() &
(lastDir.ordinal() == 0 | lastDir.ordinal() == 3))) ?
startNode : null));
            }
            lastDir = next(pointer, lastDir);
            pointer = move(lastDir, pointer);
        }
        //nový/již známý uzel
        Node endNode;
        Node returnNode;
        if (c.hasNode(pointer)) {
            endNode = c.getNode(pointer);
            returnNode = null;
        } else {
```

```
            endNode = new Node(pointer, (GFNode)
cg.get(pointer));
            c.nodes.add(endNode);
            returnNode = endNode;
        }

        Iterator<Component> it =
components.iterator();
        Component comp;
        while (it.hasNext()) {
            comp = it.next();
            if (comp.isSource) {
                if (comp.plusNode == null) {
                    comp.plusNode = endNode;
                }
            }
        }
        c.branches.add(new Branch(startNode, startDir,
endNode, lastDir.invert(), components, positions));
        //nastaví směr proudu
        ((GFNode)
cg.get(startNode.pos)).setCurrent(startDir, -1);
        ((GFNode)
cg.get(endNode.pos)).setCurrent(lastDir.invert(), 1);
        return returnNode;
    }

    private GFNode getNode(Node node) {
        return (GFNode)
cg.get(node.pos.row,node.pos.col);
    }

    /**
     * @return vstupní pozice posunutá o políčko daným
směrem
     */
    private Position move(Direction dir, Position
pointer) {
        switch (dir) {
            case NORTH:
                pointer.row--;
                break;
            case EAST:
                pointer.col++;
                break;
            case SOUTH:
                pointer.row++;
                break;
            case WEST:
                pointer.col--;
        }
        return pointer;
    }

    /**
     * @param pos pozice aktuálního pole
     * @param last výstupní směr z předchozího pole
     * @return výstupní směr z aktuáního pole
     */
    private Direction next(Position pos, Direction
last) {
        GridField gf = cg.get(pos);
        if (gf.getClass() == GFWire.class) {
            GFWire wire = (GFWire) gf;
            switch (last) {
                case NORTH:
                    return
wire.otherDir(Direction.SOUTH);
                case EAST:
                    return
wire.otherDir(Direction.WEST);
                case SOUTH:
                    return
wire.otherDir(Direction.NORTH);
                case WEST:
                    return
wire.otherDir(Direction.EAST);
            }
        }
        return last;
    }
}
```

Soubor: ./src/ApplicationLayer/Direction.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public enum Direction {

    NORTH, EAST, SOUTH, WEST;

    /**
     * @return opačný směr
     */
    public Direction invert() {
        switch (this) {
            case NORTH:
                return SOUTH;
            case SOUTH:
                return NORTH;
            case EAST:
                return WEST;
            case WEST:
                return EAST;
        }
        return null;
    }
}
```

Soubor: ./src/ApplicationLayer/EmptyLoopException.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class EmptyLoopException extends Exception {

}
```

Soubor: ./src/ApplicationLayer/Equation.java

```
package ApplicationLayer;

import ApplicationLayer.Circuit.Branch;
import ApplicationLayer.Circuit.Circuit;
import ApplicationLayer.Circuit.Loop;
import ApplicationLayer.Circuit.Node;
import java.util.ArrayList;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Equation {

    //rovnice má tvar: k1*I1+k2*I2+...+kn*In = value
    public double[] coefficients;//k1...kn
    public double value;//1.KZ: 0 2.KZ: algebraický
součet napětí ve smyčce
```

```
    /**
     * vytvoří rovnici podle 1.KZ
     */
    public Equation(Node node, Circuit c) {
        int nVariables = c.branches.size();
        ArrayList<Branch> branches = c.branches(node,
false);
        coefficients = new double[nVariables];
        Branch br;
        for (int i = 0; i < nVariables; i++) {
            br = c.branches.get(i);
            if (branches.contains(br)) {
                coefficients[i] = ((br.nodes.get(0) ==
node & br.current == -1) | ((br.nodes.get(1) == node &
br.current == 1))) ? 1 : -1;
            } else {
                coefficients[i] = 0;
            }
        }
        value = 0;
    }

    /**
     * vytvoří rovnici podle 2.KZ
     */
    public Equation(Loop loop, Circuit c) throws
ShortCircuitException {
        int nVariables = c.branches.size();
        coefficients = new double[nVariables];
        Branch br;
        value = 0;
        boolean shortCircuit = true;
        for (int i = 0; i < nVariables; i++) {
            br = c.branches.get(i);
            if (loop.branches.contains(br)) {
                coefficients[i] =
(loop.nodes.get(loop.branches.indexOf(br)) ==
br.getFirstNode()) ? br.getResistance() : -
br.getResistance();
                if (coefficients[i] != 0) {
                    shortCircuit = false;//pokud má
alespoň 1 větev odpor, není smyčka zkratovaná
                }
                value +=
(loop.nodes.get(loop.branches.indexOf(br)) ==
br.getFirstNode()) ? br.getVoltage() : -
br.getVoltage();

            } else {
                coefficients[i] = 0;
            }
        }
        value = NumOp.roundDouble(value);
        if (shortCircuit) throw new
ShortCircuitException();
    }

    @Override
    public String toString() {
        String s = "";
        double coef;
        boolean first = true;
        coef = coefficients[0];
        for (int i = 0; i < coefficients.length; i++)
{
            coef = coefficients[i];
            if (coef != 0) {
                if (coef == 1) {
                    s = (first) ?
s.concat("<i>I</i><sub>" + (i + 1) + "</sub>") :
s.concat(" + <i>I</i><sub>" + (i + 1) + "</sub>");
                } else if (coef == -1) {
                    s = s.concat(" - <i>I</i><sub>" +
(i + 1) + "</sub>");
                } else if (coef > 0) {
                    s = (first) ?
s.concat(NumOp.writeDouble(coef) + "<i>I</i><sub>" +
(i + 1) + "</sub>") : s.concat(" + " +
NumOp.writeDouble(coef) + "<i>I</i><sub>" + (i + 1) +
"</sub>");
                } else if (coef < 0) {
                    s = (first) ? s.concat("- " +
NumOp.writeDouble(-coef) + "<i>I</i><sub>" + (i + 1) +
"</sub>") : s.concat(" - " + NumOp.writeDouble(-coef)
+ "<i>I</i><sub>" + (i + 1) + "</sub>");
                }
                first = false;
            }
        }
        s = s.concat("</i> = " +
NumOp.writeDouble(value));
        //System.out.println(s);
        return s;
    }

    /*Rozpoznávání lineární ne/závislosti rovnic
    public boolean isIn(ArrayList<Equation> equations)
{
        Iterator<Equation> it = equations.iterator();
        //System.out.println("Is In " + this);
        while (it.hasNext()) {
            if (is(it.next())) {
                //System.out.println("This is same");
                return true;
            }
        }
        return false;
    }

    private boolean is(Equation e) {
        double coef;
        //System.out.println("Eq ? " + e);
        if (value == 0) {
            if (e.value != 0) {
                return false;
            } else {
                coef = e.coefficients[0] /
coefficients[0];
            }
        } else {
            coef = e.value / value;
        }
        for (int i = 0; i < coefficients.length; i++)
{
            if (coef * coefficients[i] !=
e.coefficients[i]) {
                return false;
            }
        }
        return true;
    }*/

    /**
     * @return "velikost" rovnice (počet nenulových
koeficientů)
     */
    public int getSize() {
        int size = 0;
        for (int i = 0; i < coefficients.length; i++)
{
            if (coefficients[i] != 0) {
                size++;
            }
        }
        return size;
    }
```

```
}
```

Soubor: ./src/ApplicationLayer/Equations.java

```
package ApplicationLayer;

import ApplicationLayer.Circuit.Circuit;
import ApplicationLayer.Circuit.Loop;
import ApplicationLayer.Circuit.Node;
import java.util.ArrayList;
import java.util.Iterator;
import
org.apache.commons.math.linear.Array2DRowRealMatrix;
import org.apache.commons.math.linear.ArrayRealVector;
import
org.apache.commons.math.linear.DecompositionSolver;
import
org.apache.commons.math.linear.LUDecompositionImpl;
import org.apache.commons.math.linear.RealMatrix;
import org.apache.commons.math.linear.RealVector;
import
org.apache.commons.math.linear.SingularMatrixException
;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Equations {

    public static Equations self = new Equations();
    public ArrayList<Equation> equations = new
ArrayList<Equation>();//soustava rovnic
    private double[][] coefficients;//matice
koeficientů
    private double[] values;//pole hodnot na pravé
straně rovnic
    private double[] solution;//pole řešení
    private int firstTypeEqN;

    public void generateEquations(Circuit c) throws
ShortCircuitException {
        equations.clear();
        firstTypeEqN = c.firsLawEquationsNeeded();
        int eqs = c.firsLawEquationsNeeded() +
c.secondLawEquationsNeeded();
        coefficients = new
double[eqs][c.branches.size()];
        values = new double[eqs];
        //Rce 1.KZ
        Iterator<Node> itN = c.nodes.iterator();
        Node node;
        while (itN.hasNext()) {
            node = itN.next();
            //System.out.println("\nEq Node ");
            if (equations.size() <
c.firsLawEquationsNeeded()) {
                equations.add(new Equation(node, c));
                coefficients[equations.size() - 1] =
equations.get(equations.size() - 1).coefficients;
                values[equations.size() - 1] =
equations.get(equations.size() - 1).value;
                //System.out.println("used");
            }
        }
        //Rce 2.KZ
        Iterator<Loop> itL = c.loops.iterator();
        Loop loop;
        Equation eq;
        while (itL.hasNext()) {
            loop = itL.next();
            eq = new Equation(loop, c);
            equations.add(eq);
            //System.out.println("\nEq Loop " + eq);
            coefficients[equations.size() - 1] =
equations.get(equations.size() - 1).coefficients;
            values[equations.size() - 1] =
equations.get(equations.size() - 1).value;
            //System.out.println("used");
        }
    }

    /**
     * vyřeší rovnice
     */
    public void count(Circuit c) throws
SingularMatrixException{
        RealMatrix matrix = new
Array2DRowRealMatrix(coefficients);
        //System.out.println(matrix);
        DecompositionSolver solver = new
LUDecompositionImpl(matrix).getSolver();
        RealVector constants = new
ArrayRealVector(values);
        RealVector solutions =
solver.solve(constants);
        solution = solutions.getData();
        for (int i = 0;i<solution.length;i++){
            solution[i] =
NumOp.roundDouble(solution[i]);
        }
    }

    /**
     * uloží řešení, aby bylo dostupné v popisech
prvků obvodu
     */
    public void saveSolution(Circuit c) {
        for (int i = 0; i < solution.length; i++) {
            //System.out.println("I" + (i + 1) + " = "
+ solution[i] + " = " + roundDouble(solution[i],
ACCURACY_I));

c.branches.get(i).setI(NumOp.roundDouble(solution[i]))
;
        }
    }

    /**
     * @return vrací "velikost" (počet nenulových
koeficientů) soustavy rovnic pro daný obvod (a daný
systém smyček)
     */
    public static int getPredictedSize(Circuit c)
throws ShortCircuitException {
        int size = 0;
        Iterator<Loop> itL = c.loops.iterator();
        Loop loop;
        Equation eq;
        while (itL.hasNext()) {
            loop = itL.next();
            eq = new Equation(loop, c);
            size += eq.getSize();
        }
        return size;
    }

    public double[] getSolution() {
        return solution;
    }
}
```

Soubor: ./src/ApplicationLayer/IApplicationLayer.java

```
package ApplicationLayer;
```

```
/**
 *
 * @author Jakub Hrncir (2010)
 */
public interface IApplicationLayer {

    public void run();
    public void processCG();
    public void generateEquations();
    public void solveEquations();
    public void interpretSolution();

}
```

Soubor: ./src/ApplicationLayer/IndependentCircuitsException.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class IndependentCircuitsException extends
Exception {

}
```

Soubor: ./src/ApplicationLayer/Main.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ApplicationLayer.self.run();
    }

}
```

Soubor: ./src/ApplicationLayer/NumOp.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public abstract class NumOp {//operace s čísly

    private static int ACCURACY = 4;//počet
desetinných pro výpočty

    /**
     * zaokrouhlí desetinné číslo na daný počet
desetinných míst
     */
    public static double roundDouble(double d, int
accuracy) {
        d *= Math.pow(10, accuracy + 1);
        long l = Math.round(d);
        d = (double) l / (double) 10;
        l = Math.round(d);
        d = (double) l / (double) (Math.pow(10,
accuracy));
        return d;
    }

    /**
     * zaokrouhlí desetinné číslo na nastavený počet
desetinných míst
     * pro použití při výpočtech
     */
    public static double roundDouble(double d) {
        return roundDouble(d, ACCURACY);
    }

    /**
     * @return String - zápis čísla bez desetinných 0
(příp. čárky) na konci
     */
    public static String writeDouble(double d) {
        String s = Double.toString(d);
        for (int i = s.length() - 1; i >= 0; i--) {
            if (s.charAt(i) == '0') {
                s = s.substring(0, i);
            } else if (s.charAt(i) == '.') {
                return s.substring(0, i).replace('.',
',');
            } else {
                return s.replace('.', ',');
            }
        }
        return "W";
    }

    /**
     * převede String (desetinné číslo s čárkou!) na
Double
     */
    public static Double parseDouble(String s) {
        return Double.parseDouble(s.replace(',',
'.'));
    }
}
```

Soubor: ./src/ApplicationLayer/Position.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class Position {//pozice v mřížce

    public int row;
    public int col;

    public Position(int row, int col) {
        this.row = row;
        this.col = col;
    }

    public Position(Position pos) {
        row = pos.row;
        col = pos.col;
    }

    public boolean is(Position pos) {
        return (pos.row == row & pos.col == col) ?
true : false;
    }

    public boolean is(int row, int col) {
        return (this.row == row & this.col == col) ?
true : false;
```

```
    }

    @Override
    public String toString() {
        return ("\nPosition row: " + row + " col: " +
col);
    }
}
```

Soubor: ./src/ApplicationLayer/ShortCircuitException.java

```
package ApplicationLayer;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class ShortCircuitException extends Exception {

}
```

Soubor: ./src/PresentationLayer/Forms/CircuitNotClosedException.java

```
package PresentationLayer.Forms;

/**
 * Obvod není uzavřen (obsahuje slepé větve).
 * @author Jakub Hrncir (2010)
 */
public class CircuitNotClosedException extends
Exception{

}
```

Soubor: ./src/PresentationLayer/Forms/CircuitPanel.java

```
package PresentationLayer.Forms;

import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.JEditorPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
import javax.swing.text.html.HTMLEditorKit;

/**
 * obsahuje popis prvků obvodu
 * @author Jakub Hrncir (2010)
 */
public class CircuitPanel extends JPanel {

    private MainForm mf;
    private JScrollPane scrollPane;
    private JEditorPane text;

    public CircuitPanel(MainForm mf) {
        super();
        setPreferredSize(new Dimension(250, 120));
        setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.LOWERED),"Vlastnosti prvku
obvodu",
TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,MainFo
rm.FONT_B));
        this.mf = mf;
        init();

    }

    private void init() {
        setLayout(new GridBagLayout());
        GridBagConstraints c = new
GridBagConstraints();
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1;
        c.weighty = 1;
        text = new JEditorPane();
        text.setEditable(false);
        scrollPane = new JScrollPane(text);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERT
ICAL_SCROLLBAR_ALWAYS);
        add(scrollPane, c);
        text.setEditable(false);
        text.setEditorKit(new HTMLEditorKit());
    }

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

Soubor: ./src/PresentationLayer/Forms/ComponentPanel.java

```
package PresentationLayer.Forms;

import ApplicationLayer.NumOp;
import PresentationLayer.Grid.CircuitGrid;
import PresentationLayer.Grid.GFEmpty;
import PresentationLayer.Grid.GFNode;
import PresentationLayer.Grid.GFNone;
import PresentationLayer.Grid.GFResistor;
import PresentationLayer.Grid.GFSource;
import PresentationLayer.Grid.GFWire;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

/**
 * umožňuje editovat komponenty
 * @author Jakub Hrncir (2010)
 */
public class ComponentPanel extends JPanel {

    private MainForm mf;
    private CircuitGrid cg = CircuitGrid.self;
    private JLabel lblComponent;
    private JLabel lblParameter;
    public JTextField tfValue;
```

```
    private ImageIcon confirmII =
createImageIcon("confirm.jpg");
    private JButton btnConfirm;
    private ImageIcon remII =
createImageIcon("remove.jpg");
    private JButton btnRemove;
    private ImageIcon sourceII =
createImageIcon("newSource.jpg");
    private JButton btnChangePolarity;
    private ImageIcon changePolarityII =
createImageIcon("changePolarity.jpg");
    private JButton btnSource;
    private ImageIcon resistorII =
createImageIcon("newResistor.jpg");
    private JButton btnResistor;
    private Font textEnabled = MainForm.FONT;
    private Font textDisabled = MainForm.FONT_I;

    public ComponentPanel(MainForm mf) {
        super();
        setPreferredSize(new Dimension(128, 248));
        setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.LOWERED), "Úprava
komponent", TitledBorder.LEFT,
TitledBorder.DEFAULT_POSITION, MainForm.FONT_B));
        this.mf = mf;
        init();
    }

    private void init() {
        this.setLayout(new GridBagLayout());
        GridBagConstraints c = new
GridBagConstraints();

        lblComponent = new JLabel(" ");
        c.insets = new Insets(-2, 7, 2, 0);
        c.gridwidth = 2;
        c.gridheight = 1;
        c.anchor = GridBagConstraints.NORTHWEST;
        c.fill = GridBagConstraints.NONE;
        c.gridx = 0;
        c.gridy = 0;
        c.weightx = 1;
        c.weighty = 1;
        add(lblComponent, c);

        lblParameter = new JLabel("Odpor/Napětí
(Ω/V)");
        c.insets = new Insets(0, 7, 0, 0);
        c.anchor = GridBagConstraints.WEST;
        c.gridx = 0;
        c.gridy = 1;
        add(lblParameter, c);

        tfValue = new JTextField("1");

tfValue.setHorizontalAlignment(JTextField.RIGHT);
        tfValue.setPreferredSize(new Dimension(65,
20));
        c.insets = new Insets(2, 8, 2, 0);
        c.gridx = 0;
        c.gridy = 3;
        c.weightx = 0;
        c.weighty = 1;
        add(tfValue, c);

        btnConfirm = new JButton(confirmII);
        btnConfirm.setPreferredSize(new Dimension(30,
30));
        btnConfirm.setToolTipText("Potvrdit změnu
hodnoty (Enter)");
        c.insets = new Insets(2, 8, 2, 8);
        c.fill = GridBagConstraints.NONE;
        c.ipadx = 0;
        c.anchor = GridBagConstraints.EAST;
        c.gridx = 1;
        c.gridy = 3;
        c.weightx = 1;
        c.weighty = 1;
        add(btnConfirm, c);

        btnResistor = new JButton(resistorII);
        btnResistor.setPreferredSize(new
Dimension(100, 50));
        btnResistor.setToolTipText("Přidat rezistor");
        c.anchor = GridBagConstraints.CENTER;
        c.gridx = 0;
        c.gridy = 4;
        add(btnResistor, c);

        btnSource = new JButton(sourceII);
        btnSource.setPreferredSize(new Dimension(100,
50));
        btnSource.setToolTipText("Přidat zdroj");
        c.gridx = 0;
        c.gridy = 5;
        add(btnSource, c);

        btnChangePolarity = new
JButton(changePolarityII);
        btnChangePolarity.setPreferredSize(new
Dimension(40, 40));
        btnChangePolarity.setToolTipText("Změnit
polaritu zdroje");
        c.insets = new Insets(2, 8, 2, 8);
        c.anchor = GridBagConstraints.WEST;
        c.gridx = 0;
        c.gridy = 6;
        c.gridwidth = 1;
        add(btnChangePolarity, c);

        btnRemove = new JButton(remII);
        btnRemove.setPreferredSize(new Dimension(40,
40));
        btnRemove.setToolTipText("Odebrat
komponentu");
        c.anchor = GridBagConstraints.EAST;
        c.gridx = 1;
        c.gridy = 6;
        add(btnRemove, c);

        disableAll();

        tfValue.addKeyListener(new KeyAdapter() {

            @Override
            public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() ==
KeyEvent.VK_ENTER) {
                    if (btnConfirm.isEnabled()) {
                        btnConfirm.doClick();
                    } else {
                        btnResistor.doClick();
                    }
                }
            }

            @Override
            public void keyTyped(KeyEvent e) {
                super.keyTyped(e);
                //System.out.println("Key typed: " +
e.getKeyChar());
                //System.out.println("SelSt:" +
tfValue.getSelectionStart());
                //System.out.println("SelEnd:" +
tfValue.getSelectionEnd());
                char c = e.getKeyChar();
```

```
                if (c == ',' | c == '1' | c == '2' | c
== '3' | c == '4' | c == '5' | c == '6' | c == '7' | c
== '8' | c == '9' | c == '0') {
                    int cursor =
tfValue.getCaretPosition();
                    String number = tfValue.getText();
                    int backup = number.length();
                    if (tfValue.getSelectedText() !=
null) {
                        number = number.substring(0,
tfValue.getSelectionStart()) + c +
number.substring(tfValue.getSelectionEnd());
                        if (cursor <
tfValue.getSelectionEnd()) {
                            backup -=
tfValue.getSelectedText().length();
                        }
                    } else {
                        number = number.substring(0,
cursor) + c + number.substring(cursor);
                    }
                    if (isValidN(number)) {
                        changeTFValue(number);
                    }
                    //System.out.println("Backup: " +
backup + "  Length: " + tfValue.getText().length() + "
Caret: " + cursor);
                    tfValue.setCaretPosition(cursor +
tfValue.getText().length() - backup);
                }
                e.consume();
            }
        });
        btnResistor.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                setN();
                double val =
NumOp.parseDouble(tfValue.getText());
                cg.setResistor(val);
                mf.repaintGrid();
                mf.updateP();
            }
        });

        btnSource.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                setN();
                double val =
NumOp.parseDouble(tfValue.getText());
                cg.setSource(val);
                mf.repaintGrid();
                mf.updateP();
            }
        });

        btnRemove.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                cg.removeComponent();
                mf.repaintGrid();
                mf.updateP();
            }
        });

        btnConfirm.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                setN();
                double val =
NumOp.parseDouble(tfValue.getText());
                cg.changeComponentValue(val);
                mf.repaintGrid();
            }
        });

        btnChangePolarity.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                cg.changeSourcePolarity();
                mf.repaintGrid();
            }
        });
    }

    /**
     * aktualizuje komponenty
     */
    public void updatePannel() {
        if
(cg.getSelected().getClass().equals(GFNone.class) |
cg.getSelected().getClass().equals(GFEmpty.class) |
cg.getSelected().getClass().equals(GFNode.class)) {
            disableAll();
            return;
        }
        if
(cg.getSelected().getClass().equals(GFResistor.class))
{
            disableAll();
            enableComponentEdit();
            btnChangePolarity.setEnabled(false);
            lblComponent.setText("Rezistor");
            lblParameter.setText("Odpor (Ω):");

tfValue.setText(NumOp.writeDouble(((GFResistor)
cg.getSelected()).getResistance()));
            return;
        }
        if
(cg.getSelected().getClass().equals(GFSource.class)) {
            disableAll();
            enableComponentEdit();
            btnChangePolarity.setEnabled(true);
            lblComponent.setText("Zdroj");
            lblParameter.setText("Napětí (V):");

tfValue.setText(NumOp.writeDouble(((GFSource)
cg.getSelected()).getVoltage()));
            return;
        }
        if
(cg.getSelected().getClass().equals(GFWire.class)) {
            disableAll();
            if (((GFWire)
cg.getSelected()).canPutComponent()) {
                lblComponent.setText("Zadejte
hodnotu:");
                lblComponent.setFont(textEnabled);

lblComponent.setForeground(Color.BLACK);
                btnResistor.setEnabled(true);
                btnSource.setEnabled(true);
                tfValue.setEnabled(true);
                lblParameter.setFont(textEnabled);

lblParameter.setForeground(Color.BLACK);
```

```
                lblParameter.setText("Odpor/Napětí
(Ω/V)");
            }
            return;
        }
    }

    /**
     * deaktivuje editační komponenty
     */
    private void disableComponentEdit() {
        lblComponent.setText(" ");
        lblComponent.setFont(textDisabled);
        lblComponent.setForeground(Color.GRAY);
        lblParameter.setFont(textDisabled);
        lblParameter.setForeground(Color.GRAY);
        tfValue.setEnabled(false);
        btnConfirm.setEnabled(false);
        btnRemove.setEnabled(false);
        btnChangePolarity.setEnabled(false);
    }

    /**
     * zaktivuje editační komponenty
     */
    private void enableComponentEdit() {
        lblComponent.setFont(textEnabled);
        lblComponent.setForeground(Color.BLACK);
        lblParameter.setFont(textEnabled);
        lblParameter.setForeground(Color.BLACK);
        tfValue.setEnabled(true);
        btnConfirm.setEnabled(true);
        btnRemove.setEnabled(true);
    }

    /**
     * deaktivuje celý panel
     */
    private void disableAll() {
        disableComponentEdit();
        btnResistor.setEnabled(false);
        btnSource.setEnabled(false);
    }

    private static ImageIcon createImageIcon(String
path) {
        return MainForm.createImageIcon(path);

    }

    /**
     * @return hodnota v text fieldu
     */
    public Double getComponentValue() {
        setN();
        return NumOp.parseDouble(tfValue.getText());
    }

    /**
     * testuje:
     * pouze 1 desetninná čárka
     * max 7 číslic celkem
     * max 3 číslice za desetinnou čárkou
     * @param number
     * @return odpovídá formát čísla?
     */
    private boolean isValidN(String number) {
        int comma = number.indexOf(",");
        if (comma != number.lastIndexOf(",")) {
            return false;
        }
        if (comma > -1) {
            if (comma == 7 | number.length() > 8) {
                return false;
            }
            if (number.substring(comma + 1).length() >
3) {
                return false;
            }
        } else {
            if (number.length() > 7) {
                return false;
            }
        }
        return true;
    }

    /**
     * zapíše number do tfValue
     * pokud number začíná čárkou, doplní nulu
     * pokud obsahuje zbytečné nuly na začátku,
odstraní je
     * @param number
     */
    private void changeTFValue(String number) {
        if (number.charAt(0) == ',') {
            number = "0" + number;
        }
        while (number.startsWith("00")) {
            number = number.substring(1);
        }
        tfValue.setText(number);
    }

    /**
     * zkontroluje validitu čísla v tfValue, nevhodné
hodnoty případně nahradí 1
     * poslední kontrola před využitím čísla při změně
nebo přidání komponenty
     */
    private void setN() {
        String number = tfValue.getText();
        if (isValidN(number)) {
            changeTFValue(number);
            double val =
NumOp.parseDouble(tfValue.getText());
            if (val == 0) {
                tfValue.setText("1");
            }
        } else {
            tfValue.setText("1");
        }
    }
}
```

Soubor: ./src/PresentationLayer/Forms/InfoPanel.java

```
package PresentationLayer.Forms;

import ApplicationLayer.ApplicationLayer;
import java.awt.Desktop;
import java.awt.Dimension;
import java.io.IOException;
import java.net.URISyntaxException;
import javax.swing.JEditorPane;
import javax.swing.JPanel;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;
import javax.swing.text.html.HTMLEditorKit;

/**
 * obsah pro tlačítko O Programu
 * @author Jakub Hrncir (2010)
 */
public class InfoPanel extends JPanel {
```

```
    private JEditorPane ep;

    public InfoPanel() {
        super();
        setPreferredSize(new Dimension(400, 110));
        init();
    }

    private void init() {
        ep = new JEditorPane();
        ep.setEditable(false);
        ep.setEditorKit(new HTMLEditorKit());

ep.setText(ApplicationLayer.loadFileToString("html/inf
o.html"));
        add(ep);
        ep.addHyperlinkListener(new
HyperlinkListener() {

            public void hyperlinkUpdate(HyperlinkEvent
e) {
                if (e.getEventType() ==
HyperlinkEvent.EventType.ACTIVATED) {
                    Desktop d = Desktop.getDesktop();
                    try {
                        d.browse(e.getURL().toURI());
                    } catch (IOException ex) {
                    } catch (URISyntaxException ex) {
                    }
                }
            }
        });
    }
}
```

Soubor: ./src/PresentationLayer/Forms/MainForm.java

```
package PresentationLayer.Forms;

import ApplicationLayer.ApplicationLayer;
import ApplicationLayer.Circuit.*;
import ApplicationLayer.Equation;
import PresentationLayer.Grid.CircuitGrid;
import PresentationLayer.Grid.GFNode;
import PresentationLayer.Grid.GFWire;
import PresentationLayer.GridPannel;
import PresentationLayer.IPresentationLayer;
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Desktop;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.DefaultListModel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.ListSelectionModel;
import javax.swing.ScrollPaneConstants;
import javax.swing.SwingConstants;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.text.html.HTMLEditorKit;

/**
 * Hlavní formulář aplikace
 * @author Jakub Hrncir (2010)
 */
public class MainForm extends JFrame implements
IPresentationLayer {

    private JPanel mainPanel;
    private JPanel toolPanel;
    private JPanel topPanel;
    private ResizePanel resizePanel;
    private ComponentPanel componentPanel;
    private CircuitPanel circuitPanel;
    private JButton btnNewGrid;
    private ImageIcon newGridII;
    private JButton btnZoomIn;
    private ImageIcon zoomInII;
    private JButton btnZoomOut;
    private ImageIcon zoomOutII;
    private JButton btnNewNode;
    private ImageIcon newNodeII;
    private JButton btnRemNode;
    private ImageIcon remNodeII;
    private JButton btnProcess;
    private ImageIcon processII;
    private JButton btnReturn;
    private JButton btnAbout;
    private ImageIcon returnII;
    private GridPannel gridPanel;
    private JScrollPane scrollPane;
    private JScrollPane scrollPaneMessages;
    private JEditorPane edPaneMessages;
    private JScrollPane scrollPaneEqs;
    private JEditorPane edPaneEqs;
    private JScrollPane scrollPaneSol;
    private JEditorPane edPaneSol;
    private String navod_obvod;
    private String navod_rce;
    private String header;
    private JTabbedPane tabbedPane;
    private MFState edPaneMessagesState;
    private JScrollPane scrollPaneList;
    private JSplitPane splitPane;
    private JList list;
    private JPanel listPanel;
    private DefaultListModel listModel;
    private JFrame frame;
    private int x;
    private int y;
    private CircuitGrid cg = CircuitGrid.self;
    private MFState state;
    private Circuit circuit;
    private boolean needInterpret = false;
    public static final String FONT_NAME = "Arial";
    public static final Font FONT = new
Font(FONT_NAME, Font.PLAIN, 12);
```

```
    public static final Font FONT_B = new
Font(FONT_NAME, Font.BOLD, 12);
    public static final Font FONT_I = new
Font(FONT_NAME, Font.ITALIC, 12);

    public MainForm() {
        super("Kirchhoff");
        setSize(1024, 700);
        setResizable(true);
        setMinimumSize(new Dimension(800, 600));
        setLocation(0, 0);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        state = MFState.EDITING;
        init();
        addComponentsToPane(getContentPane());
        setVisible(true);
    }

    private void init() {
        //ikony tlačítek
        newGridII = createImageIcon("newGrid.jpg");
        zoomInII = createImageIcon("zoomIn.jpg");
        zoomOutII = createImageIcon("zoomOut.jpg");
        newNodeII = createImageIcon("newNode.jpg");
        remNodeII = createImageIcon("remNode.jpg");
        processII = createImageIcon("process.jpg");
        returnII = createImageIcon("return.jpg");
        //html texty
        navod_obvod =
ApplicationLayer.loadFileToString("html/navod_obvod.ht
ml");
        navod_rce =
ApplicationLayer.loadFileToString("html/navod_rce.html
");
        header =
ApplicationLayer.loadFileToString("html/header.html");
    }

    private void addComponentsToPane(Container pane) {
        //frame pro dialogy a zprávy
        frame = new JFrame();

        //základní layout
        pane.setLayout(new BorderLayout());

        //část s obvodem
        gridPanel = new GridPannel(this);
        scrollPane = new JScrollPane(gridPanel,
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);

        //nástroje a informace vpravo - toolPanel
        toolPanel = new JPanel(new GridBagLayout());
        toolPanel.setMinimumSize(new Dimension(128,
450));
        GridBagConstraints c = new
GridBagConstraints();

        //prvky toolPanelu
        //resizePanel - změna velikosti gridu
        c.anchor =
GridBagConstraints.FIRST_LINE_START;
        c.gridx = 0;
        c.gridy = 0;
        c.weightx = 1;
        c.weighty = 0;
        resizePanel = new ResizePanel(this);
        toolPanel.add(resizePanel, c);

        //componentPanel - úprava komponent
        c.gridx = 0;
        c.gridy = 1;
        c.weightx = 0;
        c.weighty = 1;
        componentPanel = new ComponentPanel(this);
        componentPanel.setVisible(true);
        toolPanel.add(componentPanel, c);

        //circuitPanel - vlastnosti prvků obvodu
        c.gridx = 0;
        c.gridy = 2;
        c.weightx = 0;
        c.weighty = 1;
        c.fill = GridBagConstraints.BOTH;
        circuitPanel = new CircuitPanel(this);
        circuitPanel.setVisible(false);
        toolPanel.add(circuitPanel, c);

        //list částí obvodu
        listPanel = new JPanel(new BorderLayout());
        listPanel.setPreferredSize(new Dimension(260,
120));
        listPanel.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.LOWERED), "Seznam prvků
obvodu", TitledBorder.LEFT,
TitledBorder.DEFAULT_POSITION, FONT_B));
        listPanel.setVisible(false);
        listModel = new DefaultListModel();
        list = new JList(listModel);

list.setSelectionMode(ListSelectionModel.SINGLE_SELECT
ION);
        list.setFixedCellHeight(20);
        list.setFont(new Font(FONT_NAME, Font.PLAIN,
12));
        scrollPaneList = new JScrollPane(list);

scrollPaneList.setVerticalScrollBarPolicy(ScrollPaneCo
nstants.VERTICAL_SCROLLBAR_ALWAYS);
        listPanel.add(scrollPaneList);
        c.fill = GridBagConstraints.BOTH;
        c.gridx = 0;
        c.gridy = 3;
        c.weightx = 0;
        c.weighty = 1;
        toolPanel.add(listPanel, c);

        //výpisy
        edPaneMessages = new JEditorPane();
        edPaneMessages.setEditable(false);
        edPaneMessages.setEditorKit(new
HTMLEditorKit());
        edPaneMessagesState = MFState.EDITING;
        edPaneMessages.setText(navod_obvod);
        scrollPaneMessages = new
JScrollPane(edPaneMessages);
        scrollPaneMessages.setPreferredSize(new
Dimension(520, 200));

scrollPaneMessages.setVerticalScrollBarPolicy(ScrollPa
neConstants.VERTICAL_SCROLLBAR_ALWAYS);

        edPaneEqs = new JEditorPane();
        edPaneEqs.setEditable(false);
        edPaneEqs.setEditable(false);
        edPaneEqs.setEditorKit(new HTMLEditorKit());
        edPaneMessagesState = MFState.EDITING;
        scrollPaneEqs = new JScrollPane(edPaneEqs);
        scrollPaneEqs.setPreferredSize(new
Dimension(520, 200));

scrollPaneEqs.setVerticalScrollBarPolicy(ScrollPaneCon
stants.VERTICAL_SCROLLBAR_ALWAYS);

        edPaneSol = new JEditorPane();
        edPaneSol.setEditable(false);
        edPaneSol.setEditable(false);
        edPaneSol.setEditorKit(new HTMLEditorKit());
```

```
        edPaneMessagesState = MFState.EDITING;
        scrollPaneSol = new JScrollPane(edPaneSol);
        scrollPaneSol.setPreferredSize(new
Dimension(520, 200));

scrollPaneSol.setVerticalScrollBarPolicy(ScrollPaneCon
stants.VERTICAL_SCROLLBAR_ALWAYS);

        tabbedPane = new JTabbedPane(JTabbedPane.TOP);
        tabbedPane.addTab("Návod",
scrollPaneMessages);

        //nástroje nahoře topPanel
        topPanel = new JPanel(new GridBagLayout());
        topPanel.setPreferredSize(new Dimension(450,
30));

        //prvky topPanelu
        btnNewGrid = new JButton(newGridII);
        btnNewGrid.setToolTipText("Nový obvod");
        btnNewGrid.setPreferredSize(new Dimension(30,
30));
        c.fill = GridBagConstraints.NONE;
        c.anchor = GridBagConstraints.WEST;
        c.gridx = 0;
        c.gridy = 0;
        c.weightx = 0;
        c.weighty = 0;
        topPanel.add(btnNewGrid, c);

        btnZoomIn = new JButton(zoomInII);
        btnZoomIn.setToolTipText("Přiblížit");
        btnZoomIn.setPreferredSize(new Dimension(30,
30));
        c.gridx = 1;
        topPanel.add(btnZoomIn, c);

        btnZoomOut = new JButton(zoomOutII);
        btnZoomOut.setToolTipText("Oddálit");
        btnZoomOut.setPreferredSize(new Dimension(30,
30));
        c.gridx = 2;
        topPanel.add(btnZoomOut, c);

        btnNewNode = new JButton(newNodeII);
        btnNewNode.setEnabled(false);
        btnNewNode.setToolTipText("Přidat uzel");
        btnNewNode.setPreferredSize(new Dimension(30,
30));
        c.gridx = 3;
        topPanel.add(btnNewNode, c);

        btnRemNode = new JButton(remNodeII);
        btnRemNode.setEnabled(false);
        btnRemNode.setToolTipText("Odstranit uzel");
        btnRemNode.setPreferredSize(new Dimension(30,
30));
        c.gridx = 4;
        topPanel.add(btnRemNode, c);

        btnProcess = new JButton("Zpracovat obvod",
processII);
        btnProcess.setEnabled(true);
        btnProcess.setFont(FONT);
        btnProcess.setToolTipText("Pokud máte obvod
hotový, klepněte sem");

btnProcess.setHorizontalTextPosition(SwingConstants.RI
GHT);
        btnProcess.setPreferredSize(new Dimension(140,
30));
        btnProcess.setMargin(new Insets(0, -17, 0,
0));
        btnProcess.setFocusPainted(false);
        c.insets = new Insets(0, 0, 0, 0);
        c.gridx = 5;
        topPanel.add(btnProcess, c);

        btnReturn = new JButton("Změnit obvod",
returnII);
        btnReturn.setEnabled(false);
        btnReturn.setFont(FONT);

btnReturn.setHorizontalTextPosition(SwingConstants.RIG
HT);
        btnReturn.setPreferredSize(new Dimension(115,
30));
        btnReturn.setMargin(new Insets(0, -6, 0, 0));
        btnReturn.setFocusPainted(false);
        c.gridx = 6;
        topPanel.add(btnReturn, c);

        btnAbout = new JButton("O programu");
        btnAbout.setFont(FONT);
        btnAbout.setFocusPainted(false);
        c.anchor = GridBagConstraints.EAST;
        c.gridx = 7;
        c.weightx = 1;
        topPanel.add(btnAbout, c);


        //umístění základních komponent
        mainPanel = new JPanel(new BorderLayout());
        mainPanel.setPreferredSize(new Dimension(660,
400));
        mainPanel.add(scrollPane,
BorderLayout.CENTER);
        mainPanel.add(toolPanel, BorderLayout.EAST);
        splitPane = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT, mainPanel,
tabbedPane);

splitPane.getLeftComponent().setMinimumSize(new
Dimension(460, 400));

splitPane.getRightComponent().setMinimumSize(new
Dimension(200, 400));
        splitPane.setOneTouchExpandable(true);
        splitPane.setResizeWeight(1);
        splitPane.setDividerLocation(660);
        pane.add(splitPane, BorderLayout.CENTER);
        pane.add(topPanel, BorderLayout.NORTH);

        //Listenery
        addComponentListener(new ComponentListener() {

            public void
componentResized(ComponentEvent e) {
                repaintWholeGrid();
            }

            public void componentMoved(ComponentEvent
e) {
            }

            public void componentShown(ComponentEvent
e) {
            }

            public void componentHidden(ComponentEvent
e) {
            }
        });
        gridPanel.addMouseListener(new MouseListener()
{

            public void mouseClicked(MouseEvent e) {
                if (state == MFState.EDITING) {
```

```
                    gridPanel.mouseClick(e.getX(),
e.getY(), e.getButton());
                    repaintGrid();
                    updateP();
                }
            }

            public void mousePressed(MouseEvent e) {
                x = e.getX();
                y = e.getY();
            }

            public void mouseReleased(MouseEvent e) {
                if (state == MFState.EDITING) {
                    if ((x != e.getX()) | (y !=
e.getY())) {
                        gridPanel.mouseDrag(x, y,
e.getX(), e.getY(), e.getButton(), state);
                        repaintGrid();
                        updateP();
                    }
                }
            }

            public void mouseEntered(MouseEvent e) {
            }

            public void mouseExited(MouseEvent e) {
            }
        });
        gridPanel.addMouseMotionListener(new
MouseMotionListener() {

            public void mouseDragged(MouseEvent e) {
                if (state.ordinal() > 0) {
                    return;
                }
                if (gridPanel.mouseDraging(x, y,
e.getX(), e.getY())) {
                    repaintGrid();
                }
            }

            public void mouseMoved(MouseEvent e) {
            }
        });
        btnNewGrid.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                Object[] options = {"OK", "Storno"};
                int response =
JOptionPane.showOptionDialog(frame, "Nový obvod
přemaže ten stávající. Pokračovat?", "Nový obvod",
JOptionPane.OK_CANCEL_OPTION,
JOptionPane.WARNING_MESSAGE, null, options, null);
                if (response == JOptionPane.OK_OPTION)
{
                    CircuitGrid.self.resetGrid(6, 6);
                    circuitPanel.setVisible(false);
                    listPanel.setVisible(false);
                    componentPanel.setVisible(true);
                    resizePanel.setVisible(true);
                    state = MFState.EDITING;
                    repaintWholeGrid();
                    updateP();
                    listModel.clear();
                }
            }
        });
        btnZoomIn.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{

btnZoomIn.setEnabled(gridPanel.zoomIn());
                btnZoomOut.setEnabled(true);
                repaintWholeGrid();
            }
        });
        btnZoomOut.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{

btnZoomOut.setEnabled(gridPanel.zoomOut());
                btnZoomIn.setEnabled(true);
                repaintWholeGrid();
            }
        });
        btnNewNode.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                cg.setNode();
                repaintGrid();
                updateP();

            }
        });
        btnRemNode.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                cg.removeNode();
                repaintGrid();
                updateP();
            }
        });
        btnProcess.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                if (state == MFState.EDITING) {
                    try {
                        cg.canProcess();
                        cg.crop();

ApplicationLayer.self.processCG();
                        repaintGrid();
                    } catch (CircuitNotClosedException
ex) {

JOptionPane.showMessageDialog(frame, "Obvod není
uzavřený!\nOdstraňte slepé vodiče nebo doplňte ty
chybějící.", "Error", JOptionPane.ERROR_MESSAGE);
                    } catch (NoCircuitException ex) {

JOptionPane.showMessageDialog(frame, "Žádný nebo
nerozvětvený obvod!", "Error",
JOptionPane.ERROR_MESSAGE);
                    } catch (NoSourceException ex) {

JOptionPane.showMessageDialog(frame, "Obvod nemá
zdroj!", "Error", JOptionPane.ERROR_MESSAGE);
                    } catch (NoResistorException ex) {

JOptionPane.showMessageDialog(frame, "Obvod nemá žádný
odpor!", "Error", JOptionPane.ERROR_MESSAGE);
                    }
                } else if (state ==
MFState.FORMING_EQUATIONS) {
```

```
ApplicationLayer.self.generateEquations();
                } else if (state == MFState.EQUATIONS)
{

ApplicationLayer.self.solveEquations();
                } else if (state == MFState.SOLVED &
needInterpret) {

ApplicationLayer.self.interpretSolution();
                }

            }
        });
        btnReturn.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                state = MFState.EDITING;
                cg.deselect();
                cg.unHighlight();
                updateP();
                repaintGrid();
            }
        });
        btnAbout.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                JOptionPane.showMessageDialog(frame,
new InfoPanel(), "O programu Kirchhoff",
JOptionPane.PLAIN_MESSAGE);
            }
        });
        list.addListSelectionListener(new
ListSelectionListener() {

            public void
valueChanged(ListSelectionEvent e) {
                if (list.getSelectedIndex() != -1) {
                    cg.highlight((Highlightable)
listModel.getElementAt(list.getSelectedIndex()),
gridPanel.getGraphics());

circuitPanel.setText(circuit.getPartDescription(state,
list.getSelectedValue()));
                } else {
                    circuitPanel.setText("");
                }
            }
        });
        edPaneMessages.addHyperlinkListener(new
HyperlinkListener() {

            public void hyperlinkUpdate(HyperlinkEvent
e) {
                if (e.getEventType() ==
HyperlinkEvent.EventType.ACTIVATED) {
                    Desktop d = Desktop.getDesktop();
                    try {
                        d.browse(e.getURL().toURI());
                    } catch (IOException ex) {
                    } catch (URISyntaxException ex) {
                    }
                }
            }
        });
    }

    /**
     * @return hodnota editu na component pannelu
     */
    public double getComponentValue() {
        return componentPanel.getComponentValue();
    }

    /**
     * překreslí Grid
     * ! velikost musí být zachována
     */
    public void repaintGrid() {

gridPanel.paintComponent(gridPanel.getGraphics());
    }

    /**
     * překreslí Grid a přizpůsobí scrollbary
     */
    public void repaintWholeGrid() {

gridPanel.paintWholeComponent(gridPanel.getGraphics(),
scrollPane.getWidth(), scrollPane.getHeight());
        scrollPane.setViewportView(gridPanel);
    }

    /**
     * načte ikonu ze souboru
     * @param path cesta k obrázku
     * @return ImageIcon
     */
    public static ImageIcon createImageIcon(String
path) {

        try {
            InputStream is =
MainForm.class.getResourceAsStream("images/" + path);
            BufferedInputStream bis = new
BufferedInputStream(is);
            byte[] byBuf = new byte[10000];
            int byteRead = bis.read(byBuf, 0, 10000);
            ImageIcon icon = new ImageIcon(byBuf);
            return icon;
        } catch (Exception ex) {
        }

        return new ImageIcon("images/" + path);
    }

    /**
     * aktualizuje nástroje
     */
    public void updateP() {
        componentPanel.updatePannel();
        if (state == MFState.EDITING) {
            if
(cg.getSelected().getClass().equals(GFWire.class)) {
                btnNewNode.setEnabled(true);
            } else {
                btnNewNode.setEnabled(false);
            }
            if
(cg.getSelected().getClass().equals(GFNode.class) &&
!((GFNode) cg.getSelected()).needNode()) {
                btnRemNode.setEnabled(true);
            } else {
                btnRemNode.setEnabled(false);
            }
            btnProcess.setEnabled(true);
            btnProcess.setText("Zpracovat obvod");
            btnProcess.setToolTipText("Pokud máte
obvod hotový, klepněte sem");
            btnProcess.setMargin(new Insets(0, -17, 0,
0));
            btnReturn.setEnabled(false);
            circuitPanel.setVisible(false);
            listPanel.setVisible(false);
```

```
            componentPanel.setVisible(true);
            resizePanel.setVisible(true);
            if (!state.equals(edPaneMessagesState)) {
                edPaneMessagesState = MFState.EDITING;
                edPaneMessages.setText(navod_obvod);
            }
            tabbedPane.removeAll();
            tabbedPane.add("Návod",
scrollPaneMessages);
        }
        if (state.ordinal() > 0) {
            btnNewNode.setEnabled(false);
            btnRemNode.setEnabled(false);
            btnReturn.setEnabled(true);

        }
        if (state == MFState.FORMING_EQUATIONS) {
            btnProcess.setEnabled(true);
            btnProcess.setText("Ukázat rovnice");
            btnProcess.setToolTipText("Klepnutím sem
zobrazíte vytvořené rovnice");
            btnProcess.setMargin(new Insets(0, -27, 0,
0));
            if (!state.equals(edPaneMessagesState)) {
                edPaneMessagesState =
MFState.FORMING_EQUATIONS;
                edPaneMessages.setText(navod_rce);
            }
        }
        if (state == MFState.EQUATIONS) {
            btnProcess.setEnabled(true);
            btnProcess.setText("Spočítat rovnice");
            btnProcess.setToolTipText("Klepnutím sem
zobrazíte řešení rovnic");
            btnProcess.setMargin(new Insets(0, -19, 0,
0));
        }
        if (state == MFState.SOLVED) {
            if (!needInterpret) {
                btnProcess.setEnabled(false);
            } else {
                btnProcess.setText("Aktualizovat
obvod");
                btnProcess.setToolTipText("Klepnutím
sem zobrazíte výsledné směry proudu");
                btnProcess.setMargin(new Insets(0, -8,
0, 0));
            }
        }
        if (state == MFState.INTERPRETED) {
            btnProcess.setEnabled(false);
        }
    }

    /**
     * zobrazí prvky obvodu, přejde k vytváření rovnic
     * @param c
     */
    public void showBranches(Circuit c) {
        circuit = c;
        circuitPanel.setVisible(true);
        listPanel.setVisible(true);
        listModel.clear();
        componentPanel.setVisible(false);
        edPaneMessagesState =
MFState.FORMING_EQUATIONS;
        edPaneMessages.setText(navod_rce);
        resizePanel.setVisible(false);
        cg.deselect();
        state = MFState.FORMING_EQUATIONS;
        repaintGrid();
        updateP();
        Iterator<Node> itN = c.nodes.iterator();
        while (itN.hasNext()) {
            listModel.addElement(itN.next());
        }
        Iterator<Branch> itB = c.branches.iterator();
        while (itB.hasNext()) {
            listModel.addElement(itB.next());
        }
        Iterator<Loop> itL = c.loops.iterator();
        while (itL.hasNext()) {
            listModel.addElement(itL.next());
        }
        showMessages();
    }

    /**
     * zobrazí vytvořené rovnice
     * @param equations
     */
    public void showEquations(ArrayList<Equation>
equations) {
        list.clearSelection();
        cg.unHighlight();
        repaintGrid();
        state = MFState.EQUATIONS;
        tabbedPane.addTab("Rovnice", scrollPaneEqs);

edPaneEqs.setText(getEquationsText(equations));
        tabbedPane.setSelectedIndex(1);
        updateP();
        showMessages();
    }

    /**
     * zobrazí řešení
     * @param solution
     * @param needInterpret
     */
    public void showSolution(double[] solution,
boolean needInterpret) {
        this.needInterpret = needInterpret;
        list.clearSelection();
        cg.unHighlight();
        repaintGrid();
        tabbedPane.addTab("Řešení", scrollPaneSol);
        edPaneSol.setText(getSolutionsText(solution));
        tabbedPane.setSelectedIndex(2);
        state = MFState.SOLVED;
        updateP();
        showMessages();
    }

    /**
     * zajistí viditelnost části s texty
(nápověda,rovnice,řešení) a viditelnost mřížky
     */
    private void showMessages() {
        int dl = splitPane.getDividerLocation();
        //System.out.println("Divider: " + dl + "
width:" + splitPane.getWidth());
        if (dl > splitPane.getWidth() - 30) {
            splitPane.setDividerLocation(660);
        }
        if (dl < 461) {
            splitPane.setDividerLocation(461);
        }
    }

    /**
     * interpretuje řešení
     */
    public void interpretSolution() {
        state = MFState.INTERPRETED;
        list.clearSelection();
        repaintGrid();
        updateP();
```

```
    }

    /**
     * zobrazí chybovou hlášku
     * @param message
     */
    public void showErrorMessage(String message) {
        JOptionPane.showMessageDialog(frame, message,
"Error", JOptionPane.ERROR_MESSAGE);
    }

    /**
     * @return je potřeba zobrazovat oznaceni uzlu a
vetvi?
     */
    public boolean needMarks() {
        if (state.ordinal() == 0) {
            return false;
        }
        return true;
    }

    /**
     * vygeneruje text pro vytvořené rovnice
     * @param equations
     * @return
     */
    private String
getEquationsText(ArrayList<Equation> equations) {
        String s = "";
        int first = circuit.firsLawEquationsNeeded();
        s = s.concat(header);
        s = s.concat("<h2>Vytvořené
rovnice</h2><h3>První Kirchhoffův zákon</h3>");
        for (int i = 0; i < first; i++) {
            s = s.concat("<p>" +
equations.get(i).toString() + "</p>");
        }
        s = s.concat("<h3>Druhý Kirchhoffův
zákon</h3>");
        for (int i = first; i < equations.size(); i++)
{
            s = s.concat("<p>" +
equations.get(i).toString() + "</p>");
        }
        return s;
    }

    /**
     * vygeneruje text pro vypočtená řešení
     * @param solution
     * @return
     */
    private String getSolutionsText(double[] solution)
{
        String s = "";
        int branches = circuit.branches.size();
        boolean zapor = false;
        s = s.concat(header);
        s = s.concat("<h2>Vypočtené hodnoty
proudu</h2>");
        for (int i = 0; i < branches; i++) {
            if (solution[i] < 0) {
                zapor = true;
            }
            s = s.concat("<p> <i>I</i><sub>" + (i + 1)
+ "</sub> = " + String.format("%.4f", solution[i]) + "
A</p>");
        }
        if (zapor) {
            s = s.concat("<p>Záporné hodnoty u
některých velikostí proudů znamenají, že proud danou
větví proudí ve skutečnosti opačným směrem, než jsem
zvolili. Jeho směr je tedy opačný, než jak ukazují
šipky v nákresu.</p>");
                s = s.concat("<p>Pokud zvolené směry
proudu nesouhlasí s výslednými, můžete jejich směry v
nákresu změnit klepnutím na tlačítko
<strong>Aktualizovat obvod</strong> (budou upraveny i
příslušné hodnoty proudů v popisech prvků
obvodu).</p>");
        }
        s = s.concat("<p>Pokud je hodnota proudu nula,
proud touto větví vůbec neteče, nebo je menší než 0,1
mA.</p>");
        return s;
    }
}
```

Soubor: ./src/PresentationLayer/Forms/MFState.java

```
package PresentationLayer.Forms;

/**
 * fáze, ve kterých se může nacházet program
 * @author Jakub Hrncir (2010)
 */
public enum MFState {
    EDITING, FORMING_EQUATIONS, EQUATIONS, SOLVED,
INTERPRETED;

}
```

Soubor: ./src/PresentationLayer/Forms/NoCircuitException.java

```
package PresentationLayer.Forms;

/**
 * Žádný nebo nerozvětvený obvod (neobsahuje žádný
uzel).
 * @author Jakub Hrncir (2010)
 */
public class NoCircuitException extends Exception{

}
```

Soubor: ./src/PresentationLayer/Forms/NoResistorException.java

```
package PresentationLayer.Forms;

/**
 * Obvod nemá žádný rezistor.
 * @author Jakub Hrncir (2010)
 */
public class NoResistorException extends Exception {

}
```

Soubor: ./src/PresentationLayer/Forms/NoSourceException.java

```
package PresentationLayer.Forms;

/**
 * Obvod nemá žádný zdroj.
```

```
 * @author Jakub Hrncir (2010)
 */
public class NoSourceException extends Exception {

}
```

Soubor: ./src/PresentationLayer/Forms/ResizePanel.java

```
package PresentationLayer.Forms;

import ApplicationLayer.Direction;
import PresentationLayer.Grid.CircuitGrid;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

/**
 * Panel umožňující změnu velikosti mřížky.
 * @author Jakub Hrncir (2010)
 */
public class ResizePanel extends JPanel {

    private MainForm mf;
    private JButton addNorth;
    private JButton addEast;
    private JButton addSouth;
    private JButton addWest;
    private JButton cropGrid;
    private ImageIcon anII =
createImageIcon("addNorth.jpg");
    private ImageIcon aeII =
createImageIcon("addEast.jpg");
    private ImageIcon asII =
createImageIcon("addSouth.jpg");
    private ImageIcon awII =
createImageIcon("addWest.jpg");
    private ImageIcon crII =
createImageIcon("cropGrid.jpg");
    private int extensionPower = 2;

    public ResizePanel(MainForm mf) {
        super();
        setPreferredSize(new Dimension(128, 142));
        setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.LOWERED),"Velikost mřížky",
TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,MainFo
rm.FONT_B));
        this.mf = mf;
        init();
    }

    private void init() {
        this.setLayout(new GridBagLayout());
        GridBagConstraints c = new
GridBagConstraints();
        addNorth = new JButton(anII);
        addNorth.setPreferredSize(new Dimension(36,
36));
        addNorth.setToolTipText("Přidá 2 řádky mřížky
nahoře");
        addEast = new JButton(aeII);
        addEast.setPreferredSize(new Dimension(36,
36));
        addEast.setToolTipText("Přidá 2 sloupce mřížky
vpravo");
        addSouth = new JButton(asII);
        addSouth.setPreferredSize(new Dimension(36,
36));
        addSouth.setToolTipText("Přidá 2 řádky mřížky
dole");
        addWest = new JButton(awII);
        addWest.setPreferredSize(new Dimension(36,
36));
        addWest.setToolTipText("Přidá 2 sloupce mřížky
vlevo");
        cropGrid = new JButton(crII);
        cropGrid.setPreferredSize(new Dimension(36,
36));
        cropGrid.setToolTipText("Oříznout mřížku");

        c.anchor = GridBagConstraints.NORTH;
        c.insets = new Insets(1, 1, 1, 1);
        c.gridwidth = 1;
        c.gridx = 1;
        c.gridy = 0;
        c.weightx = 0;
        c.weighty = 0;
        add(addNorth, c);
        c.gridx = 0;
        c.gridy = 1;
        c.weightx = 0;
        c.weighty = 0;
        add(addWest, c);
        c.gridx = 1;
        c.gridy = 1;
        c.weightx = 0;
        c.weighty = 0;
        add(cropGrid, c);
        c.gridx = 2;
        c.gridy = 1;
        c.weightx = 0.1;
        c.weighty = 0;
        add(addEast, c);
        c.gridx = 1;
        c.gridy = 2;
        c.weightx = 0;
        c.weighty = 0.1;
        add(addSouth, c);

        addNorth.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{

CircuitGrid.self.extend(Direction.NORTH,
extensionPower);
                mf.repaintWholeGrid();
            }
        });
        addEast.addActionListener(new ActionListener()
{

            public void actionPerformed(ActionEvent e)
{

CircuitGrid.self.extend(Direction.EAST,
extensionPower);
                mf.repaintWholeGrid();
            }
        });
        addSouth.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
```

```
CircuitGrid.self.extend(Direction.SOUTH,
extensionPower);
                mf.repaintWholeGrid();
            }
        });
        addWest.addActionListener(new ActionListener()
{

            public void actionPerformed(ActionEvent e)
{

CircuitGrid.self.extend(Direction.WEST,
extensionPower);
                mf.repaintWholeGrid();
            }
        });
        cropGrid.addActionListener(new
ActionListener() {

            public void actionPerformed(ActionEvent e)
{
                CircuitGrid.self.crop();
                mf.repaintWholeGrid();
                mf.updateP();
            }
        });
    }

    protected static ImageIcon createImageIcon(String
path) {
        return MainForm.createImageIcon(path);
    }
}
```

Soubor: ./src/PresentationLayer/Grid/CircuitGrid.java

```
package PresentationLayer.Grid;

import ApplicationLayer.Circuit.Highlightable;
import ApplicationLayer.Circuit.Loop;
import ApplicationLayer.Position;
import ApplicationLayer.Direction;
import
PresentationLayer.Forms.CircuitNotClosedException;
import PresentationLayer.Forms.NoCircuitException;
import PresentationLayer.Forms.NoResistorException;
import PresentationLayer.Forms.NoSourceException;
import PresentationLayer.GridPannel;
import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import java.awt.image.BufferedImage;
import java.util.Iterator;

/**
 * obsahuje data mřížky
 * @author Jakub Hrncir (2010)
 */
public class CircuitGrid {

    public static CircuitGrid self = new
CircuitGrid(6, 6);
    private static GFNone none = new GFNone();
    private ArrayList<CircuitGridLine> gridlines;
    private int GRIDCONSTANT = 60;//aktuální velikost
polička
    private int MINGRIDSIZE = 3;//minimální počet
sloupců a řádků
    private int MINFIELDSIZE = 40;//minimální velikost
polička
    private int MAXFIELDSIZE = 150;//maximální
velikost polička
    private GridField lastDragSelected = none;
    private GridField selected = none;
    private int selectedX;
    private int selectedY;
    public GridPannel gp;

    private CircuitGrid(int row, int col) {
        gridlines = new ArrayList<CircuitGridLine>();
        selected = none;
        for (int i = 0; i < row; i++) {
            gridlines.add(new CircuitGridLine(col));
        }
    }

    public void resetGrid(int row, int col) {
        gridlines.clear();
        for (int i = 0; i < row; i++) {
            gridlines.add(new CircuitGridLine(col));
        }
        selected = none;
    }

    /**
     * přidá dráty do prázdných políček a do políček s
dráty a uzly
     * přidá uzly, pokud je to nutné (tzn 3 větve v
poli)
     * @param where řádek/sloupec(rozlišeno par.
horizontal), kam se přidává drát
     * @param from políčko, kde drát začíná
     * @param to políčko, kde drát končí TO>FROM
     * @param horizontal drát je vodorovný/svislý
     */
    public void drawWire(int where, int from, int to,
boolean horizontal) {
        if (horizontal) {
            GridField gf = get(where, from);
            if (gf.getClass().equals(GFEmpty.class)) {
                set(where, from, new GFWire(false,
true, false, false));
            } else if
(gf.getClass().equals(GFWire.class) |
gf.getClass().equals(GFNode.class)) {
                ((GFWire) gf).setEast(true);
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(where, from, new
GFNode((GFWire) gf));
                }
            }
            for (int i = 1; i < to - from; i++) {
                gf = get(where, from + i);
                if
(gf.getClass().equals(GFEmpty.class)) {
                    set(where, from + i, new
GFWire(false, true, false, true));
                }
                if (gf.getClass().equals(GFWire.class)
| gf.getClass().equals(GFNode.class)) {
                    ((GFWire) gf).setEast(true);
                    ((GFWire) gf).setWest(true);
                    if
(gf.getClass().equals(GFWire.class) & ((GFWire)
gf).needNode()) {
                        set(where, from + i, new
GFNode((GFWire) gf));
                    }
                }
            }
            gf = get(where, to);
            if (gf.getClass().equals(GFEmpty.class)) {
```

```
                set(where, to, new GFWire(false,
false, false, true));
            } else if
(gf.getClass().equals(GFWire.class) |
gf.getClass().equals(GFNode.class)) {
                ((GFWire) gf).setWest(true);
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(where, to, new GFNode((GFWire)
gf));
                }
            }
        } else {
            GridField gf = get(from, where);
            if (gf.getClass().equals(GFEmpty.class)) {
                set(from, where, new GFWire(false,
false, true, false));
            } else if
(gf.getClass().equals(GFWire.class) |
gf.getClass().equals(GFNode.class)) {
                ((GFWire) gf).setSouth(true);
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(from, where, new
GFNode((GFWire) gf));
                }
            }
            for (int i = 1; i < to - from; i++) {
                gf = get(from + i, where);
                if
(gf.getClass().equals(GFEmpty.class)) {
                    set(from + i, where, new
GFWire(true, false, true, false));
                }
                if (gf.getClass().equals(GFWire.class)
| gf.getClass().equals(GFNode.class)) {
                    ((GFWire) gf).setNorth(true);
                    ((GFWire) gf).setSouth(true);
                    if
(gf.getClass().equals(GFWire.class) & ((GFWire)
gf).needNode()) {
                        set(from + i, where, new
GFNode((GFWire) gf));
                    }
                }
            }
            gf = get(to, where);
            if (gf.getClass().equals(GFEmpty.class)) {
                set(to, where, new GFWire(true, false,
false, false));
            } else if
(gf.getClass().equals(GFWire.class) |
gf.getClass().equals(GFNode.class)) {
                ((GFWire) gf).setNorth(true);
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(to, where, new GFNode((GFWire)
gf));
                }
            }
        }
    }

    /**
     * odstraní dráty a uzly
     * @param where řádek/sloupec(rozlišeno par.
horizontal), kde se odebírá drát
     * @param from políčko, kde mazání začíná
     * @param to políčko, kde mazání končí TO>FROM
     * @param horizontal mazaný drát je
vodorovný/svislý
     */
    public void deleteWire(int where, int from, int
to, boolean horizontal) {
        GridField gf;
        if (horizontal) {
            gf = get(where, from);
            if (gf.getClass() == GFWire.class |
gf.getClass() == GFNode.class) {
                ((GFWire) gf).setEast(false);
                if (gf.getClass() == GFNode.class &&
!((GFNode) gf).needNode()) {
                    set(where, from, new
GFWire((GFNode) gf));
                }
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(where, from, new
GFNode((GFWire) gf));
                }
            }
            if (gf.getClass() == GFSource.class |
gf.getClass() == GFResistor.class) {
                if (((Orientable) gf).isHorizontal())
{
                    set(where, from, new GFEmpty());
                }
            }
            for (int i = 1; i < to - from; i++) {
                gf = get(where, from + i);
                if (gf.getClass() == GFWire.class |
gf.getClass() == GFNode.class) {
                    ((GFWire) gf).setEast(false);
                    ((GFWire) gf).setWest(false);
                    if (gf.getClass() == GFNode.class
&& !((GFNode) gf).needNode()) {
                        set(where, from + i, new
GFWire((GFNode) gf));
                    }
                }
                if (gf.getClass() == GFSource.class |
gf.getClass() == GFResistor.class) {
                    if (((Orientable)
gf).isHorizontal()) {
                        set(where, from + i, new
GFEmpty());
                    }
                }
            }
            gf = get(where, to);
            if (gf.getClass() == GFWire.class |
gf.getClass() == GFNode.class) {
                ((GFWire) gf).setWest(false);
                if (gf.getClass() == GFNode.class &&
!((GFNode) gf).needNode()) {
                    set(where, to, new GFWire((GFNode)
gf));
                }
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(where, to, new GFNode((GFWire)
gf));
                }
            }
            if (gf.getClass() == GFSource.class |
gf.getClass() == GFResistor.class) {
                if (((Orientable) gf).isHorizontal())
{
                    set(where, to, new GFEmpty());
                }
            }
        } else {
            gf = get(from, where);
            if (gf.getClass() == GFWire.class |
gf.getClass() == GFNode.class) {
                ((GFWire) gf).setSouth(false);
                if (gf.getClass() == GFNode.class &&
!((GFNode) gf).needNode()) {
```

```
                    set(from, where, new
GFWire((GFNode) gf));
                }
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(from, where, new
GFNode((GFWire) gf));
                }
            }
            if (gf.getClass() == GFSource.class |
gf.getClass() == GFResistor.class) {
                if (!((Orientable) gf).isHorizontal())
{
                    set(from, where, new GFEmpty());
                }
            }
            for (int i = 1; i < to - from; i++) {
                gf = get(from + i, where);
                if (gf.getClass() == GFWire.class |
gf.getClass() == GFNode.class) {
                    ((GFWire) gf).setNorth(false);
                    ((GFWire) gf).setSouth(false);
                    if (gf.getClass() == GFNode.class
&& !((GFNode) gf).needNode()) {
                        set(from + i, where, new
GFWire((GFNode) gf));
                    }
                }
                if (gf.getClass() == GFSource.class |
gf.getClass() == GFResistor.class) {
                    if (!((Orientable)
gf).isHorizontal()) {
                        set(from + i, where, new
GFEmpty());
                    }
                }
            }
            gf = get(to, where);
            if (gf.getClass() == GFWire.class |
gf.getClass() == GFNode.class) {
                ((GFWire) gf).setNorth(false);
                if (gf.getClass() == GFNode.class &&
!((GFNode) gf).needNode()) {
                    set(to, where, new GFWire((GFNode)
gf));
                }
                if (gf.getClass().equals(GFWire.class)
& ((GFWire) gf).needNode()) {
                    set(to, where, new GFNode((GFWire)
gf));
                }
            }
            if (gf.getClass() == GFSource.class |
gf.getClass() == GFResistor.class) {
                if (!((Orientable) gf).isHorizontal())
{
                    set(to, where, new GFEmpty());
                }
            }
        }
        clearEmptyWires();
    }

    /**
     * označí políčka jako "označená tažením"
     * @param where řádek/sloupec(rozlišeno par.
horizontal), odkud se označuje
     * @param from políčko, kde mazání začíná
     * @param to políčko, kde mazání končí TO>FROM
     * @param horizontal mazaný drát je
vodorovný/svislý
     */
    public void dragSelect(int where, int from, int
to, boolean horizontal) {
        deselect();
        if (horizontal) {
            for (int i = 0; i < to - from + 1; i++) {
                get(where, from +
i).setDragSelected(true);
            }
        } else {
            for (int i = 0; i < to - from + 1; i++) {
                get(from + i,
where).setDragSelected(true);
            }
        }
    }

    /**
     * rozšíří grid
     * @param dir směr rozšíření
     * @param number počet nových řádků/sloupců
     */
    public void extend(Direction dir, int number) {
        if (dir.equals(Direction.NORTH)) {
            for (int i = 0; i < number; i++) {
                gridlines.add(0, new
CircuitGridLine(getColCount()));
            }
        }
        if (dir == Direction.EAST) {
            for (int i = 0; i < number; i++) {
                for (int r = 0; r < getRowCount();
r++) {
                    gridlines.get(r).fields.add(new
GFEmpty());
                }
            }
        }
        if (dir == Direction.SOUTH) {
            for (int i = 0; i < number; i++) {
                gridlines.add(new
CircuitGridLine(getColCount()));
            }
        }
        if (dir == Direction.WEST) {
            for (int i = 0; i < number; i++) {
                for (int r = 0; r < getRowCount();
r++) {
                    gridlines.get(r).fields.add(0, new
GFEmpty());
                }
            }
        }
    }

    /**
     * ořízne grid o prázdné sloupce a řádky, až do
minimálního rozměru, přednostně shora a zleva
     */
    public void crop() {
        selected = none;
        clearEmptyWires();
        while (getRowCount() > MINGRIDSIZE &
isEmpty(Direction.NORTH)) {
            gridlines.remove(0);
        }
        while (getRowCount() > MINGRIDSIZE &
isEmpty(Direction.SOUTH)) {
            gridlines.remove(getRowCount() - 1);
        }
        while (getColCount() > MINGRIDSIZE &
isEmpty(Direction.WEST)) {
            for (int i = 0; i < getRowCount(); i++) {
                gridlines.get(i).fields.remove(0);
            }
        }
```

```
        while (getColCount() > MINGRIDSIZE &
isEmpty(Direction.EAST)) {
            int formerColCount = getColCount();
            for (int i = 0; i < getRowCount(); i++) {

gridlines.get(i).fields.remove(formerColCount - 1);
            }
        }
    }

    /**
     * nahradí GFEmpty místo GFWires (a GFNodes) bez
drátů (north,east,south a west == false)
     */
    private void clearEmptyWires() {
        for (int i = 0; i < getRowCount(); i++) {
            gridlines.get(i).clearEmptyWires();
        }
    }

    /**
     * zjistí, zda je daný sloupec/řádek prázdný
     * @param dir směr, ze kterého se vlastnost
zjišťuje
     */
    private boolean isEmpty(Direction dir) {
        if (dir == Direction.NORTH) {
            return gridlines.get(0).isEmpty();
        }
        if (dir == Direction.SOUTH) {
            return gridlines.get(getRowCount() -
1).isEmpty();
        }
        if (dir == Direction.EAST) {
            for (int i = 0; i < getRowCount(); i++) {
                if
(!gridlines.get(i).fields.get(getColCount() -
1).getClass().equals(GFEmpty.class)) {
                    return false;
                }
            }
        }
        if (dir == Direction.WEST) {
            for (int i = 0; i < getRowCount(); i++) {
                if
(!gridlines.get(i).fields.get(0).getClass().equals(GFE
mpty.class)) {
                    return false;
                }
            }
        }
        return true;
    }

    /**
     * @return šířka mřížky v px
     */
    public int getWidth() {
        return getColCount() * getGRIDCONSTANT();
    }

    /**
     * @return výška mřížky v px
     */
    public int getHeight() {
        return getRowCount() * getGRIDCONSTANT();
    }

    /**
     * @return počet řádků
     */
    public int getRowCount() {
        return gridlines.size();
    }

    /**
     * @return počet sloupců
     */
    public int getColCount() {
        return gridlines.get(0).getColCount();
    }

    /**
     * @param row řádek
     * @param col sloupec
     * @return field na dané pozici
     */
    public GridField get(int row, int col) {
        return gridlines.get(row).fields.get(col);
    }

    /**
     * @param pos
     * @return field na dané pozici
     */
    public GridField get(Position pos) {
        return get(pos.row, pos.col);
    }

    /**
     * nahradí určený field novým
     * @param row řádek
     * @param col sloupec
     * @param field nový field
     */
    private void set(int row, int col, GridField
field) {
        gridlines.get(row).fields.set(col, field);
    }

    /**
     * překreslí Grid do dodaného grafického kontextu
pomocí Double Bufferingu
     * @param paintMarks vykreslit označení větví a
uzlů?
     */
    public void paint(Graphics g, boolean paintMarks)
{
        BufferedImage buf = new
BufferedImage(gp.getWidth(), gp.getHeight(),
BufferedImage.TYPE_INT_RGB);
        Graphics offG = buf.getGraphics();
        offG.setColor(Color.GRAY);
        offG.fillRect(0, 0, buf.getWidth(),
buf.getHeight());
        for (int row = 0; row < getRowCount(); row++)
{
            for (int col = 0; col < getColCount();
col++) {
                boolean b = get(row,
col).equals(getSelected());
                /*if (b) {
                System.out.print("T");
                } else {
                System.out.print("_");
                }*/
                get(row, col).paint(offG, col *
getGRIDCONSTANT(), row * getGRIDCONSTANT(),
getGRIDCONSTANT(), (get(row,
col).equals(getSelected())), paintMarks);
            }
            //System.out.println("");
        }
        //System.out.println("");
        g.drawImage(buf, 0, 0, null);
    }

    /**
```

```
     * zruší veškeré označení políček gridu
     */
    public void deselect() {
        setSelected(none);
        deDragSelect();
    }

    /**
     * zruší označení políček tažením (modré)
     */
    public void deDragSelect() {
        for (int c = 0; c < getColCount(); c++) {
            for (int r = 0; r < getRowCount(); r++) {
                get(r, c).setDragSelected(false);
            }
        }
    }

    /**
     * označí všechna pole jako nezvýrazněná
     */
    public void unHighlight() {
        for (int c = 0; c < getColCount(); c++) {
            for (int r = 0; r < getRowCount(); r++) {
                get(r, c).setHighlighted(false);
            }
        }
    }

    /**
     * nahradí označený úsek drátu zdrojem
     * @param voltage napětí nového zdroje
     */
    public void setSource(double voltage) {
        GFSource source = new GFSource((GFWire)
getSelected(), voltage);
        set(selectedY, selectedX, source);
        setSelected(source);
    }

    /**
     * nahradí označený úsek drátu rezistorem
     * @param resistance odpor zdroje
     */
    public void setResistor(double resistance) {
        GFResistor resistor = new GFResistor((GFWire)
getSelected(), resistance);
        set(selectedY, selectedX, resistor);
        setSelected(resistor);
    }

    /**
     * přidá na označené políčko uzel
     */
    public void setNode() {
        GFNode node = new GFNode((GFWire)
getSelected());
        set(selectedY, selectedX, node);
        setSelected(node);
    }

    /**
     * odstraní uzel s označeného políčka
     */
    public void removeNode() {
        GFWire wire = new GFWire((GFNode)
getSelected());
        set(selectedY, selectedX, wire);
        setSelected(wire);
    }

    /**
     * nahradí označenou komponentu rovným úsekem
drátu
     */
    public void removeComponent() {
        if (getSelected().getClass() ==
GFResistor.class) {
            GFWire wire = new GFWire((GFResistor)
getSelected());
            set(selectedY, selectedX, wire);
            setSelected(wire);
        }
        if (getSelected().getClass() ==
GFSource.class) {
            GFWire wire = new GFWire((GFSource)
getSelected());
            set(selectedY, selectedX, wire);
            setSelected(wire);
        }

    }

    /**
     * změní napětí/odpor označeného zdroje/rezistoru
     * @param value nová hodnota komponenty
     */
    public void changeComponentValue(double value) {
        if (getSelected().getClass() ==
GFResistor.class) {
            ((GFResistor)
getSelected()).setResistance(value);
        }
        if (getSelected().getClass() ==
GFSource.class) {
            ((GFSource)
getSelected()).setVoltage(value);
        }

    }

    /**
     * zmeni polaritu zdroje
     */
    public void changeSourcePolarity() {
        ((GFSource) getSelected()).changePolarity();
    }

    /**
     * označí příslušnou část obvodu za zvýrazněnou a
přidělí polím barvy zvýraznění
     * @param part zvýrazněná část obvodu
     */
    public void highlight(Highlightable part, Graphics
g) {
        deselect();
        unHighlight();
        Iterator<Position> it =
part.getPositions().iterator();
        Position pos;
        double green = 0;
        double greenInc = 0;
        boolean loop = false;
        if (part.getClass() == Loop.class) {
            //System.out.println("\nLoop
Highlighted");
            loop = true;
            double d = part.getPositions().size() - 1;
            greenInc = 1 / d;

//System.out.println("GreenInc:"+greenInc);
        }
        while (it.hasNext()) {
            pos = it.next();
            GridField field = get(pos.row, pos.col);
            field.setHighlighted(true);
            field.setHighlightColor();
            //System.out.println(field);
```

```
            if (loop) {
                Color c =
field.setHighlightColor(green);
                //System.out.println("Highlight
position "+pos.row+" "+pos.col);
                //System.out.println("\n"+ green);
                //System.out.println("Color:"+c);
                green += greenInc;
            }
        }
        paint(g, true);
    }

    /**
     * označí uzly po řádkách písmeny A, B, C...
     */
    public void setMarks() {
        GridField gf;
        char c = 'A';
        for (int row = 0; row < getRowCount(); row++)
{
            for (int col = 0; col < getColCount();
col++) {
                gf = get(row, col);
                if
(gf.getClass().equals(GFNode.class)) {
                    ((GFNode)
gf).setMark(Character.toString(c));
                    c++;
                }
            }
        }
    }

    /**
     * @return the GRIDCONSTANT
     */
    public int getGRIDCONSTANT() {
        return GRIDCONSTANT;
    }

    /**
     * @param GRIDCONSTANT the GRIDCONSTANT to set
     */
    public void setGRIDCONSTANT(int GRIDCONSTANT) {
        this.GRIDCONSTANT = GRIDCONSTANT;
    }

    /**
     * @return the lastDragSelected
     */
    public GridField getLastDragSelected() {
        return lastDragSelected;
    }

    /**
     * @param lastDragSelected the lastDragSelected to
set
     */
    public void setLastDragSelected(GridField
lastDragSelected) {
        this.lastDragSelected = lastDragSelected;
    }

    /**
     * @return the selected
     */
    public GridField getSelected() {
        return selected;
    }

    /**
     * @param selected the selected to set
     */
    public void setSelected(GridField selected) {
        this.selected = selected;
    }

    /**
     * @param selectedX the selectedX to set
     */
    public void setSelectedX(int selectedX) {
        this.selectedX = selectedX;
    }

    /**
     * @param selectedY the selectedY to set
     */
    public void setSelectedY(int selectedY) {
        this.selectedY = selectedY;
    }

    /**
     * @return the MINFIELDSIZE
     */
    public int getMINFIELDSIZE() {
        return MINFIELDSIZE;
    }

    /**
     * zkontroluje zpracovatelnost mřížky, při
nalezení chyby hodí výjimku
     */
    public void canProcess() throws
CircuitNotClosedException, NoCircuitException,
NoSourceException, NoResistorException {
        GridField gf;
        int sources = 0;
        int nodes = 0;
        int resistors = 0;
        for (int row = 0; row < getRowCount(); row++)
{
            for (int col = 0; col < getColCount();
col++) {
                gf = get(row, col);
                if
(gf.getClass().equals(GFWire.class)) {
                    if (((GFWire) gf).numberOfWires()
== 1) {
                        throw new
CircuitNotClosedException();
                    }
                }
                if
(gf.getClass().equals(GFSource.class)) {
                    sources++;
                }
                if
(gf.getClass().equals(GFResistor.class)) {
                    resistors++;
                }
                if
(gf.getClass().equals(GFNode.class)) {
                    ((GFNode) gf).resetCurrents();
                    nodes++;
                }
            }
        }
        if (nodes == 0) {
            throw new NoCircuitException();
        }
        if (sources == 0) {
            throw new NoSourceException();
        }
        if (resistors == 0) {
            throw new NoResistorException();
        }
    }
```

```
    /**
     * @return the MAXFIELDSIZE
     */
    public int getMAXFIELDSIZE() {
        return MAXFIELDSIZE;
    }
}
```

Soubor: ./src/PresentationLayer/Grid/CircuitGridLine.java

```
package PresentationLayer.Grid;

import java.util.ArrayList;

/**
 * jeden řádek mřížky
 * @author Jakub Hrncir (2010)
 */
public class CircuitGridLine {

    public ArrayList<GridField> fields;

    public CircuitGridLine(int col) {
        fields = new ArrayList<GridField>();
        for (int i = 0; i < col; i++) {
            fields.add(new GFEmpty());
        }
    }

    /**
     * @return zda je řádek prázdný
     */
    public boolean isEmpty() {
        for (int i = 0; i < fields.size(); i++) {
            if
(!fields.get(i).getClass().equals(GFEmpty.class)) {
                return false;
            }
        }
        return true;
    }

    /**
     * nahradí GFEmpty místo GFWires (a GFNodes) bez
drátů (north,east,south a west == false)
     */
    public void clearEmptyWires() {
        for (int i = 0; i < fields.size(); i++) {
            if (fields.get(i).isCleared()) {
                fields.set(i, new GFEmpty());
            }
        }
    }

    public int getColCount(){
        return fields.size();
    }
}
```

Soubor: ./src/PresentationLayer/Grid/GFEmpty.java

```
package PresentationLayer.Grid;

import java.awt.Color;
import java.awt.Graphics;

/**
 * prázdné políčko
 * @author Jakub Hrncir (2010)
 */
public class GFEmpty extends GridField {

    /**
     * nastaví barvu zvýraznění
     * @param green musí být mezi 0-1, 0 je
nejsvětlejší, 1 nejtmavší zelená
     */
    @Override
    public Color setHighlightColor(double green) {
        color_highlight = new Color((int) ((1 - green)
* COLOR_CONST), 255, (int) ((1 - green) *
COLOR_CONST));
        return new Color((int) ((1 - green) *
COLOR_CONST), 255, (int) ((1 - green) * COLOR_CONST));
    }

    @Override
    public void paint(Graphics g, int x1, int y1, int
GC, boolean selected, boolean paintMarks) {
        g.setColor((selected) ? COLOR_SELECT :
COLOR_NOSELECT);
        if (dragSelected) {
            g.setColor(COLOR_DRAGSELECT);
        }
        if (highlighted) {
            g.setColor(color_highlight);
        }
        g.fillRect(x1, y1, GC, GC);
        g.setColor(COLOR_BORDER);
        g.drawRect(x1, y1, GC, GC);

    }

    @Override
    public boolean isCleared() {
        return false;
    }

    @Override
    public void setHighlightColor() {
        color_highlight = COLOR_HIGHLIGHT;
    }
}
```

Soubor: ./src/PresentationLayer/Grid/GFNode.java

```
package PresentationLayer.Grid;

import ApplicationLayer.Direction;
import java.awt.Font;
import java.awt.Graphics;

/**
 * políčko uzlu
 * @author Jakub Hrncir (2010)
 */
public class GFNode extends GFWire {

    private boolean knowCurrents;
    private int currentNorth;//1:směr proudu je k
uzlu/-1: od uzlu/0 nevede proud
    private int currentEast;
    private int currentSouth;
    private int currentWest;
    private String mark = "";
    public int[] branches = new int[4];

    public GFNode(GFWire wire) {
        super(wire.north, wire.east, wire.south,
wire.west);
        knowCurrents = false;
```

```
    }

    @Override
    public void paint(Graphics g, int x1, int y1, int
GC, boolean selected, boolean paintMarks) {
        g.setFont(new Font("Arial", Font.PLAIN, GC /
5));
        g.setColor((selected) ? COLOR_SELECT :
COLOR_NOSELECT);
        setFillColor(g, selected);
        g.fillRect(x1, y1, GC, GC);
        g.setColor(COLOR_BORDER);
        g.drawRect(x1, y1, GC, GC);
        g.setColor(COLOR_LINE);
        if (isNorth()) {
            g.drawLine(x1 + (GC / 2), y1, x1 + (GC /
2), y1 + (GC / 2));
            if (paintMarks & knowCurrents) {
                if (currentNorth == 1) {
                    g.drawLine(x1 + (2 * GC / 5), y1 +
(3 * GC / 10), x1 + (GC / 2), y1 + (2 * GC / 5));
                    g.drawLine(x1 + (3 * GC / 5), y1 +
(3 * GC / 10), x1 + (GC / 2), y1 + (2 * GC / 5));
                }
                if (currentNorth == -1) {
                    g.drawLine(x1 + (2 * GC / 5), y1 +
GC / 10, x1 + (GC / 2), y1);
                    g.drawLine(x1 + (3 * GC / 5), y1 +
GC / 10, x1 + (GC / 2), y1);
                }
            }
            //g.setColor(COLOR_BORDER);
            if (paintMarks & branches[0] != 0) {
                setFillColor(g, selected);
                g.fillRect(x1 + 7 * GC / 20, y1 + 11 *
GC / 40 - GC / 6, (int)
Math.round(Integer.toString(branches[0]).length() * GC
/ 9), GC / 6 + 1);
                g.setColor(COLOR_LINE);

g.drawString(Integer.toString(branches[0]), x1 + 7 *
GC / 20, y1 + 11 * GC / 40);
            }
        }
        g.setColor(COLOR_LINE);
        if (isEast()) {
            g.drawLine(x1 + (GC / 2), y1 + (GC / 2),
x1 + GC, y1 + (GC / 2));
            if (paintMarks & knowCurrents) {
                if (currentEast == 1) {
                    g.drawLine(x1 + 7 * GC / 10, y1 +
(2 * GC / 5), x1 + (3 * GC / 5), y1 + (GC / 2));
                    g.drawLine(x1 + 7 * GC / 10, y1 +
(3 * GC / 5), x1 + (3 * GC / 5), y1 + (GC / 2));
                }
                if (currentEast == -1) {
                    g.drawLine(x1 + 9 * GC / 10, y1 +
(2 * GC / 5), x1 + GC, y1 + (GC / 2));
                    g.drawLine(x1 + 9 * GC / 10, y1 +
(3 * GC / 5), x1 + GC, y1 + (GC / 2));
                }
            }
            if (paintMarks & branches[1] != 0) {

g.drawString(Integer.toString(branches[1]), x1 + 3 *
GC / 4, y1 + 2 * GC / 5);
            }
        }
        g.setColor(COLOR_LINE);
        if (isSouth()) {
            g.drawLine(x1 + (GC / 2), y1 + (GC / 2),
x1 + (GC / 2), y1 + GC);
            if (paintMarks & knowCurrents) {
                if (currentSouth == 1) {
                    g.drawLine(x1 + (2 * GC / 5), y1 +
7 * GC / 10, x1 + (GC / 2), y1 + (3 * GC / 5));
                    g.drawLine(x1 + (3 * GC / 5), y1 +
7 * GC / 10, x1 + (GC / 2), y1 + (3 * GC / 5));
                }
                if (currentSouth == -1) {
                    g.drawLine(x1 + (2 * GC / 5), y1 +
9 * GC / 10, x1 + (GC / 2), y1 + GC);
                    g.drawLine(x1 + (3 * GC / 5), y1 +
9 * GC / 10, x1 + (GC / 2), y1 + GC);
                }
            }
            if (paintMarks & branches[2] != 0) {
                setFillColor(g, selected);
                g.fillRect(x1 + 7 * GC / 20, y1 + 7 *
GC / 8 - GC / 6, (int)
Math.round(Integer.toString(branches[2]).length() * GC
/ 9), GC / 6 + 1);
                g.setColor(COLOR_LINE);

g.drawString(Integer.toString(branches[2]), x1 + 7 *
GC / 20, y1 + 7 * GC / 8);
            }
        }
        g.setColor(COLOR_LINE);
        if (isWest()) {
            g.drawLine(x1, y1 + (GC / 2), x1 + (GC /
2), y1 + (GC / 2));
            if (paintMarks & knowCurrents) {
                if (currentWest == 1) {
                    g.drawLine(x1 + 3 * GC / 10, y1 +
(2 * GC / 5), x1 + (2 * GC / 5), y1 + (GC / 2));
                    g.drawLine(x1 + 3 * GC / 10, y1 +
(3 * GC / 5), x1 + (2 * GC / 5), y1 + (GC / 2));
                }
                if (currentWest == -1) {
                    g.drawLine(x1 + GC / 10, y1 + (2 *
GC / 5), x1, y1 + (GC / 2));
                    g.drawLine(x1 + GC / 10, y1 + (3 *
GC / 5), x1, y1 + (GC / 2));
                }
            }
            if (paintMarks & branches[3] != 0) {

g.drawString(Integer.toString(branches[3]), x1 + 3 *
GC / 20, y1 + 2 * GC / 5);
            }
        }
        g.fillOval((int) (x1 + 0.4 * GC), (int) (y1 +
0.4 * GC), (int) (0.2 * GC), (int) (0.2 * GC));
        g.setColor(COLOR_BORDER);
        if (paintMarks) {
            g.drawString(getMark(), x1 + (17 * GC /
20), y1 + (39 * GC / 40));
        }
    }

    /**
     * @param dir
     * @param current 1/0/-1 → k uzlu/žádný/od uzlu
     */
    public void setCurrent(Direction dir, int current)
{
        switch (dir) {
            case NORTH:
                currentNorth = current;
                break;
            case EAST:
                currentEast = current;
                break;
            case SOUTH:
                currentSouth = current;
                break;
            case WEST:
```

```
            currentWest = current;
        }
    }

    public void resetCurrents() {
        currentNorth = 0;
        currentEast = 0;
        currentSouth = 0;
        currentWest = 0;

    }

    public int getCurrent(Direction dir) {
        switch (dir) {
            case NORTH:
                return currentNorth;
            case EAST:
                return currentEast;
            case SOUTH:
                return currentSouth;
            case WEST:
                return currentWest;
            default:
                return 0;
        }
    }

    public void invertCurrent(Direction dir) {
        switch (dir) {
            case NORTH:
                currentNorth = -currentNorth;
                break;
            case EAST:
                currentEast = -currentEast;
                break;
            case SOUTH:
                currentSouth = -currentSouth;
                break;
            case WEST:
                currentWest = -currentWest;
        }
    }

    /**
     * @param know if field knows the currents
     */
    public void setKnowCurrents(boolean know) {
        knowCurrents = know;
    }

    /**
     * @param mark the mark to set
     */
    public void setMark(String mark) {
        this.mark = mark;
    }

    /**
     * @return the mark
     */
    public String getMark() {
        return mark;
    }

    public void setBranch(Direction dir, int mark) {
        switch (dir) {
            case NORTH:
                branches[0] = mark;
                break;
            case EAST:
                branches[1] = mark;
                break;
            case SOUTH:
                branches[2] = mark;
                break;
            case WEST:
                branches[3] = mark;
            }
        }

        public char getCharMark() {
            return mark.charAt(0);
        }
    }
```

Soubor: ./src/PresentationLayer/Grid/GFNone.java

```
package PresentationLayer.Grid;

import java.awt.Color;
import java.awt.Graphics;

/**
 * "žádné políčko" - aby proměnná selected nebyla
null, ale šlo odlišit stav, kdy není žádné políčko
označené
 * @author Jakub Hrncir (2010)
 */
public class GFNone extends GridField {

    @Override
    public void paint(Graphics g, int x1, int y1, int
GC, boolean selected, boolean paintMarks) {
    }

    @Override
    public boolean isCleared() {
        return false;
    }

    @Override
    public Color setHighlightColor(double green) {
        return null;
    }
    @Override
    public void setHighlightColor() {
    }

}
```

Soubor: ./src/PresentationLayer/Grid/GFResistor.java

```
package PresentationLayer.Grid;

import ApplicationLayer.NumOp;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;

/**
 * políčko rezistoru
 * @author Jakub Hrncir (2010)
 */
public class GFResistor extends GridField implements
Orientable{

    private boolean horizontal;
    private double resistance;

    public GFResistor(GFWire wire, double resistance)
{
        horizontal = (wire.isHorizontal()) ? true :
false;
```

```
        this.resistance = resistance;
    }
    /**
     * nastaví barvu zvýraznění
     * @param green musí být mezi 0-1, 0 je
nejsvětlejší, 1 nejtmavší zelená
     */
    @Override
    public Color setHighlightColor(double green) {
        color_highlight = new Color((int) ((1 - green)
* COLOR_CONST), 255, (int) ((1 - green) *
COLOR_CONST));
        return new Color((int) ((1 - green) *
COLOR_CONST), 255, (int) ((1 - green) * COLOR_CONST));
    }
    @Override
    public void setHighlightColor() {
        color_highlight = COLOR_HIGHLIGHT;
    }

    @Override
    public void paint(Graphics g, int x1, int y1, int
GC, boolean selected, boolean paintMarks) {
        setFillColor(g, selected);
        g.fillRect(x1, y1, GC, GC);
        g.setColor(COLOR_BORDER);
        g.drawRect(x1, y1, GC, GC);
        g.setColor(COLOR_LINE);
        if (isHorizontal()) {
            g.drawLine(x1, y1 + (GC / 2), x1 + (GC /
4), y1 + (GC / 2));
            g.drawRect(x1 + (GC / 4), y1 + 3 * (GC /
8), GC / 2, GC / 4);
            g.drawLine(x1 + GC - (GC / 4), y1 + (GC /
2), x1 + GC, y1 + (GC / 2));
            g.setFont(new Font("Arial", Font.PLAIN, GC
/ 5));

g.drawString(NumOp.writeDouble(resistance), x1 + GC /
8, y1 + GC / 4);
        } else {
            g.drawLine(x1 + (GC / 2), y1, x1 + (GC /
2), y1 + (GC / 4));
            g.drawRect(x1 + 3 * (GC / 8), y1 + (GC /
4), GC / 4, GC / 2);
            g.drawLine(x1 + (GC / 2), y1 + GC - (GC /
4), x1 + (GC / 2), y1 + GC);
            g.setFont(new Font("Arial", Font.PLAIN, GC
/ 5));
            setFillColor(g, selected);
            g.fillRect(x1 + GC / 8, y1 + (2 * GC / 5),
(int)Math.round((NumOp.writeDouble(resistance).length(
)-0.2)*GC/9), GC / 4);
            g.setColor(COLOR_LINE);

g.drawString(NumOp.writeDouble(resistance), x1 + GC /
8, y1 + (3 * GC / 5));
        }
    }

    @Override
    public boolean isCleared() {
        return false;
    }

    /**
     * @return the resistance
     */
    public double getResistance() {
        return resistance;
    }

    /**
     * @param resistance the resistance to set
     */
    public void setResistance(double resistance) {
        this.resistance = resistance;
    }

    /**
     * @return the horizontal
     */
    public boolean isHorizontal() {
        return horizontal;
    }

}
```

Soubor: ./src/PresentationLayer/Grid/GFSource.java

```
package PresentationLayer.Grid;

import ApplicationLayer.NumOp;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;

/**
 * políčko zdroje
 * @author Jakub Hrncir (2010)
 */
public class GFSource extends GridField implements
Orientable{

    private boolean horizontal;
    private double voltage;
    private boolean polarity;

    public GFSource(GFWire wire, double voltage) {
        horizontal = (wire.isHorizontal()) ? true :
false;
        polarity = true;
        this.voltage = voltage;
    }

    @Override
    public void paint(Graphics g, int x1, int y1, int
GC, boolean selected, boolean paintMarks) {
        g.setColor((selected) ? COLOR_SELECT :
COLOR_NOSELECT);
        if (dragSelected) {
            g.setColor(COLOR_DRAGSELECT);
        }
        if (highlighted) {
            g.setColor(color_highlight);
        }
        g.fillRect(x1, y1, GC, GC);
        g.setColor(COLOR_BORDER);
        g.drawRect(x1, y1, GC, GC);
        g.setColor(COLOR_LINE);
        if (isHorizontal()) {
            g.drawLine(x1, y1 + (GC / 2), x1 + (GC /
4), y1 + (GC / 2));
            g.drawOval(x1 + GC / 4, y1 + 7 * GC / 16,
GC / 8, GC / 8);
            g.drawOval(x1 + GC - GC / 4 - GC / 8, y1 +
7 * GC / 16, GC / 8, GC / 8);
            g.drawLine(x1 + GC - (GC / 4), y1 + (GC /
2), x1 + GC, y1 + (GC / 2));
            g.setFont(new Font("Arial", Font.PLAIN, GC
/ 5));
            g.drawString(NumOp.writeDouble(voltage),
x1 + GC / 8, y1 + GC / 4);
            g.setFont(new Font("Arial", Font.PLAIN, GC
/ 3));
```

```
            g.drawString((getPolarity()) ? "+" : "-",
x1 + GC / 5, y1 + 4 * GC / 5);
            g.drawString((getPolarity()) ? "-" : "+",
x1 + 3 * GC / 5, y1 + 4 * GC / 5);
        } else {
            g.drawLine(x1 + (GC / 2), y1, x1 + (GC /
2), y1 + (GC / 4));
            g.drawOval(x1 + 7 * GC / 16, y1 + GC / 4,
GC / 8, GC / 8);
            g.drawOval(x1 + 7 * GC / 16, y1 + GC - GC
/ 4 - GC / 8, GC / 8, GC / 8);
            g.drawLine(x1 + (GC / 2), y1 + GC - (GC /
4), x1 + (GC / 2), y1 + GC);
            g.setFont(new Font("Arial", Font.PLAIN, GC
/ 5));
            g.setColor(COLOR_LINE);
            g.drawString(NumOp.writeDouble(voltage),
x1 + GC / 8, y1 + (23 * GC / 40));
            g.setFont(new Font("Arial", Font.PLAIN, GC
/ 3));
            g.drawString((getPolarity()) ? "+" : "-",
x1 + GC / 5, y1 + 1 * GC / 3);
            g.drawString((getPolarity()) ? "-" : "+",
x1 + GC / 5, y1 + 6 * GC / 7);
        }
    }

    @Override
    public boolean isCleared() {
        return false;
    }
    /**
     * nastaví barvu zvýraznění
     * @param green musí být mezi 0-1, 0 je
nejsvětlejší, 1 nejtmavší zelená
     */
    @Override
    public Color setHighlightColor(double green) {
        color_highlight = new Color((int) ((1 - green)
* COLOR_CONST), 255, (int) ((1 - green) *
COLOR_CONST));
        return new Color((int) ((1 - green) *
COLOR_CONST), 255, (int) ((1 - green) * COLOR_CONST));
    }
    @Override
    public void setHighlightColor() {
        color_highlight = COLOR_HIGHLIGHT;
    }

    /**
     * @return the voltage
     */
    public double getVoltage() {
        return voltage;
    }

    /**
     * @param voltage the voltage to set
     */
    public void setVoltage(double voltage) {
        this.voltage = voltage;
    }

    /**
     * @return the horizontal
     */
    public boolean isHorizontal() {
        return horizontal;
    }

    /**
     * @return the polarity
     */
    public boolean getPolarity() {
        return polarity;
    }

    public void changePolarity() {
        polarity = !polarity;
    }
}
```

Soubor: ./src/PresentationLayer/Grid/GFWire.java

```
package PresentationLayer.Grid;

import ApplicationLayer.Direction;
import java.awt.Graphics;
import java.util.ArrayList;

/**
 * políčko prázdného vodiče
 * @author Jakub Hrncir (2010)
 */
public class GFWire extends GridField {

    protected boolean north;
    protected boolean east;
    protected boolean south;
    protected boolean west;

    public GFWire(boolean north, boolean east, boolean
south, boolean west) {
        this.north = north;
        this.east = east;
        this.south = south;
        this.west = west;
    }

    public GFWire(GFNode node) {
        this(node.north, node.east, node.south,
node.west);
    }

    public GFWire(GFResistor resistor) {
        this(!(resistor.isHorizontal()),
(resistor.isHorizontal()), !(resistor.isHorizontal()),
(resistor.isHorizontal()));
    }

    public GFWire(GFSource source) {
        this(!(source.isHorizontal()),
(source.isHorizontal()), !(source.isHorizontal()),
(source.isHorizontal()));
    }

    @Override
    public void paint(Graphics g, int x1, int y1, int
GC, boolean selected, boolean paintMarks) {
        g.setColor((selected) ? COLOR_SELECT :
COLOR_NOSELECT);
        if (dragSelected) {
            g.setColor(COLOR_DRAGSELECT);
        }
        if (highlighted) {
            g.setColor(color_highlight);
        }
        g.fillRect(x1, y1, GC, GC);
        g.setColor(COLOR_BORDER);
        g.drawRect(x1, y1, GC, GC);
        g.setColor(COLOR_LINE);
        if (isNorth()) {
            g.drawLine(x1 + (GC / 2), y1, x1 + (GC /
2), y1 + (GC / 2));
        }
        if (isEast()) {
```

```
            g.drawLine(x1 + (GC / 2), y1 + (GC / 2),
x1 + GC, y1 + (GC / 2));
        }
        if (isSouth()) {
            g.drawLine(x1 + (GC / 2), y1 + (GC / 2),
x1 + (GC / 2), y1 + GC);
        }
        if (isWest()) {
            g.drawLine(x1, y1 + (GC / 2), x1 + (GC /
2), y1 + (GC / 2));
        }
    }

    public boolean isCleared() {
        if (north) {
            return false;
        }
        if (east) {
            return false;
        }
        if (south) {
            return false;
        }
        if (west) {
            return false;
        }
        return true;
    }

    public boolean isHorizontal() {
        return (west == true & east == true & north ==
false & south == false);
    }

    public boolean isVertical() {
        return (west == false & east == false & north
== true & south == true);
    }

    /**
     *
     * @return true pokud lze umístit componentu
(jedná se o rovný úsek drátu, nekříží se)
     */
    public boolean canPutComponent() {
        return (isHorizontal() | isVertical());
    }

    protected int numberOfWires() {
        int i = 0;
        if (north) {
            i++;
        }
        if (east) {
            i++;
        }
        if (south) {
            i++;
        }
        if (west) {
            i++;
        }
        return i;
    }

    public boolean needNode() {
        return (numberOfWires() == 3);
    }

    public Direction otherDir(Direction dir) {
        ArrayList<Direction> dirs = getDirs();
        if (dirs.size() == 4) {
            return dir.invert();
        }
        if (dirs.size() != 2) {
            System.out.println("otherDir Error
(GFWire)");
            return null;
        }
        return (dirs.get(0).ordinal() ==
dir.ordinal()) ? dirs.get(1) : dirs.get(0);
    }

    public ArrayList<Direction> getDirs() {
        ArrayList<Direction> dirs = new
ArrayList<Direction>(4);
        if (north) {
            dirs.add(Direction.NORTH);
        }
        if (east) {
            dirs.add(Direction.EAST);
        }
        if (south) {
            dirs.add(Direction.SOUTH);
        }
        if (west) {
            dirs.add(Direction.WEST);
        }
        return dirs;
    }

    protected boolean isDirection(Direction dir) {
        if (dir == Direction.NORTH & north == true) {
            return true;
        }
        if (dir == Direction.EAST & east == true) {
            return true;
        }
        if (dir == Direction.SOUTH & south == true) {
            return true;
        }
        if (dir == Direction.WEST & west == true) {
            return true;
        }
        return false;
    }

    /**
     * @return the north
     */
    public boolean isNorth() {
        return north;
    }

    /**
     * @param north the north to set
     */
    public void setNorth(boolean north) {
        this.north = north;
    }

    /**
     * @return the east
     */
    public boolean isEast() {
        return east;
    }

    /**
     * @param east the east to set
     */
    public void setEast(boolean east) {
        this.east = east;
    }

    /**
     * @return the south
     */
```

```
    public boolean isSouth() {
        return south;
    }

    /**
     * @param south the south to set
     */
    public void setSouth(boolean south) {
        this.south = south;
    }

    /**
     * @return the west
     */
    public boolean isWest() {
        return west;
    }

    /**
     * @param west the west to set
     */
    public void setWest(boolean west) {
        this.west = west;
    }
}
```

Soubor: ./src/PresentationLayer/Grid/GridField.java

```
package PresentationLayer.Grid;

import java.awt.Color;
import java.awt.Graphics;

/**
 * políčko mřížky
 * @author Jakub Hrncir (2010)
 */
public abstract class GridField {

    protected boolean dragSelected = false;
    protected boolean highlighted = false;
    protected Color COLOR_NOSELECT = Color.WHITE;
    protected Color COLOR_SELECT = Color.YELLOW;
    protected Color COLOR_DRAGSELECT = new Color(153,
255, 255);
    protected Color COLOR_LINE = Color.BLACK;
    protected Color COLOR_BORDER = Color.GRAY;
    protected Color COLOR_HIGHLIGHT = Color.GREEN;
    protected Color color_highlight = COLOR_HIGHLIGHT;
    protected int COLOR_CONST = 220;

    public abstract void paint(Graphics g, int x1, int
y1, int GC, boolean selected, boolean paintMarks);

    public void setDragSelected(boolean selected) {
        dragSelected = selected;
    }

    public void setHighlighted(boolean highlighted) {
        this.highlighted = highlighted;
    }

    public abstract boolean isCleared();

    /**
     * nastaví barvu zvýraznění
     * @param green musí být mezi 0-1, 0 je
nejsvětlejší, 1 nejtmavší zelená
     */
    public Color setHighlightColor(double green) {
        color_highlight = new Color((int) ((1 - green)
* COLOR_CONST), 255, (int) ((1 - green) *
COLOR_CONST));
        return new Color((int) ((1 - green) *
COLOR_CONST), 255, (int) ((1 - green) * COLOR_CONST));
    }

    /**
     * nastaví barvu zvýraznění na standartní
(nejtmavší)
     */
    public void setHighlightColor() {
        color_highlight = COLOR_HIGHLIGHT;
    }

    protected void setFillColor(Graphics g, boolean
selected) {
        g.setColor((selected) ? COLOR_SELECT :
COLOR_NOSELECT);
        if (dragSelected) {
            g.setColor(COLOR_DRAGSELECT);
        }
        if (highlighted) {
            g.setColor(color_highlight);
        }
    }
}
```

Soubor: ./src/PresentationLayer/Grid/Orientable.java

```
package PresentationLayer.Grid;

/**
 * umožňuje porovnávat otočení zdrojů a rezistorů
 * @author Jakub Hrncir (2010)
 */
interface Orientable {
    public boolean isHorizontal();

}
```

Soubor: ./src/PresentationLayer/GridPannel.java

```
package PresentationLayer;

import PresentationLayer.Forms.MFState;
import PresentationLayer.Forms.MainForm;
import PresentationLayer.Grid.CircuitGrid;
import PresentationLayer.Grid.GFResistor;
import PresentationLayer.Grid.GFSource;
import PresentationLayer.Grid.GFWire;
import PresentationLayer.Grid.GridField;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JPanel;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public class GridPannel extends JPanel {

    private CircuitGrid cg;
    private MainForm mf;

    /**
     * panel obsahující nákres obvodu
     */
    public GridPannel(MainForm mf) {
        this.mf = mf;
        cg = CircuitGrid.self;
```

```
    }

    /**
     * vykreslí mřížku i její okolí
     * @param width šířka scrollpane
     * @param height výška scrollpane
     */
    public void paintWholeComponent(Graphics g, int
width, int height) {
        cg.gp = this;
        setPreferredSize(new Dimension(cg.getWidth(),
cg.getHeight()));
        paintComponent(g);
    }

    /**
     * vykreslí mřížku
     * ! rozměry musí být zachovány
     */
    @Override
    public void paintComponent(Graphics g) {
        //super.paintComponent(g); toto způsobuje
blikání
        cg.paint(g, mf.needMarks());
    }

    /**
     * ošetří kliknutí
     * @param x mouse click x (px)
     * @param y mouse click y (px)
     */
    public void mouseClick(int x, int y, int
mouseButton) {
        int GC = cg.getGRIDCONSTANT();
        if ((x < cg.getWidth()) & (y <
cg.getHeight())) {
            //System.out.println("Click");
            GridField gf = cg.get(y / GC, x / GC);
            if (!gf.equals(cg.getSelected())) {
                cg.setSelected(cg.get(y / GC, x /
GC));
                cg.setSelectedX(x / GC);
                cg.setSelectedY(y / GC);
            } else {
                if (mouseButton == 3) {
                    gf = cg.getSelected();
                    if (gf.getClass() == GFWire.class)
{
                        if (((GFWire)
gf).canPutComponent()) {

cg.setResistor(mf.getComponentValue());
                        }
                    }
                    if (gf.getClass() ==
GFResistor.class) {
                        cg.removeComponent();

cg.setSource(mf.getComponentValue());
                    }
                    if (gf.getClass() ==
GFSource.class) {
                        cg.removeComponent();
                    }
                }
            }
            //System.out.println("sel: " + (y / GC) +
" " + (x / GC));
        }
    }

    /**
     * ošetří mouse drag (drag musí skončit buď ve
stejném řádku, nebo sloupci)
     * pokud drag skončí ve stejné buňce, vyhodnotí se
jako kliknutí
     * @param x1 mouse pressed x (px)
     * @param y1 mouse pressed y (px)
     * @param x2 mouse released x (px)
     * @param y2 mouse released y (px)
     */
    public void mouseDrag(int x1, int y1, int x2, int
y2, int button, MFState state) {
        int GC = cg.getGRIDCONSTANT();
        if ((x1 < cg.getWidth()) & (y1 <
cg.getHeight()) & (x2 < cg.getWidth()) & (y2 <
cg.getHeight())) {
            if ((y1 / GC == y2 / GC) & (x1 / GC == x2
/ GC)) {
                if (button == 1 | button == 3) {
                    cg.deDragSelect();
                    mouseClick(x2, y2, button);
                    return;
                }
            }
            if (state.ordinal() > 0) {
                return;
            }
            if ((y1 / GC == y2 / GC) & (x1 / GC != x2
/ GC)) {
                if (button == 1) {
                    cg.drawWire(y1 / GC, Math.min(x1 /
GC, x2 / GC), Math.max(x1 / GC, x2 / GC), true);
                }
                if (button == 3) {
                    cg.deleteWire(y1 / GC, Math.min(x1
/ GC, x2 / GC), Math.max(x1 / GC, x2 / GC), true);
                }

            }
            if ((y1 / GC != y2 / GC) & (x1 / GC == x2
/ GC)) {
                if (button == 1) {
                    cg.drawWire(x1 / GC, Math.min(y1 /
GC, y2 / GC), Math.max(y1 / GC, y2 / GC), false);
                }
                if (button == 3) {
                    cg.deleteWire(x1 / GC, Math.min(y1
/ GC, y2 / GC), Math.max(y1 / GC, y2 / GC), false);
                }
            }
        }
        cg.deselect();
    }

    /**
     * ošetří probíhající mouse dragging (drag musí
být buď ve stejném řádku, nebo sloupci)
     * @param x1 mouse pressed x (px)
     * @param y1 mouse pressed y (px)
     * @param x2 mouse now x (px)
     * @param y2 mouse now y (px)
     * return zda se změnil vzhled gridu (zda je
potřeba vykreslení)
     */
    public boolean mouseDraging(int x1, int y1, int
x2, int y2) {
        boolean changed = false;
        int GC = cg.getGRIDCONSTANT();
        if ((x1 < cg.getWidth()) & (y1 <
cg.getHeight()) & (x2 < cg.getWidth()) & (y2 <
cg.getHeight())) {
            if ((y1 / GC == y2 / GC) & (x1 / GC != x2
/ GC) & !(cg.get(y2 / GC, x2 /
GC).equals(cg.getLastDragSelected()))) {
                cg.dragSelect(y1 / GC, Math.min(x1 /
GC, x2 / GC), Math.max(x1 / GC, x2 / GC), true);
```

```
                cg.setLastDragSelected(cg.get(y2 / GC,
x2 / GC));
                changed = true;
            }
            if ((y1 / GC != y2 / GC) & (x1 / GC == x2
/ GC) & !(cg.get(y2 / GC, x2 /
GC).equals(cg.getLastDragSelected()))) {
                cg.dragSelect(x1 / GC, Math.min(y1 /
GC, y2 / GC), Math.max(y1 / GC, y2 / GC), false);
                cg.setLastDragSelected(cg.get(y2 / GC,
x2 / GC));
                changed = true;
            }
        }
        return changed;
    }

    /**
     * ošetří zvětšení
     * @return má být povoleno další použití této
metody
     */
    public boolean zoomIn() {
        if (cg.getGRIDCONSTANT() <
cg.getMAXFIELDSIZE()) {
            cg.setGRIDCONSTANT(cg.getGRIDCONSTANT() +
10);
        }
        return (cg.getGRIDCONSTANT() <
cg.getMAXFIELDSIZE()) ? true : false;
    }

    /**
     * ošetří zmenšení
     * @return má být povoleno další použití této
metody
     */
    public boolean zoomOut() {
        if (cg.getGRIDCONSTANT() >
cg.getMINFIELDSIZE()) {
            cg.setGRIDCONSTANT(cg.getGRIDCONSTANT() -
10);
        }
        return (cg.getGRIDCONSTANT() >
cg.getMINFIELDSIZE()) ? true : false;
    }
}
```

Soubor: ./src/PresentationLayer/IPresentationLayer.java

```
package PresentationLayer;

import ApplicationLayer.Circuit.Circuit;
import ApplicationLayer.Equation;
import java.util.ArrayList;

/**
 *
 * @author Jakub Hrncir (2010)
 */
public interface IPresentationLayer {

    public void showBranches(Circuit c);
    public void showEquations(ArrayList<Equation>
equations);
    public void showSolution(double[] solution,
boolean needInterpret);
    public void interpretSolution();
    public void showErrorMessage(String message);
}
```