ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

BAKALÁŘSKÁ PRÁCE

Knihovna pro vykreslování grafů

Martin Voráček **2016/2017**

Originál (kopie) zadání BP/DP

Abstrakt

Bakalářská práce je zaměřena na možnosti a použití grafické knihovny SWT a zdůvodnění proč zvolit právě knihovnu SWT, pro použití při vykreslování grafů a věcí s tím související. Zároveň i pomocí této knihovny navrhnout a vypracovat knihovnu s podrobným postupem, rozborem jak to vše funguje, zapadá do sebe a jak je navržen systémem tříd. Následně vytvořit sadu několika příkladů tříd, které budou implementovat různé typy grafů například graf koláčový, sloupcový, bodový,... Dále pak vytvořit několik příkladů použití.

Klíčová slova

Widget, Shell, Display, Java, Knihovna, Rámec, Grafický kontext, Graf, Swing, Standard Widget Toolkit, Abstract Windowing Toolkit, SWTChar, Eclipse.

Abstract

The bachelor´s thesis is focused on the possibilities of using the SWT graphic library and the explanation why to choose such this SWT library for the depiction of graphs and closely connected things. It also explains how to design and work with this library in the similar use, the detailed analysis how everything goes, falls together and how the system of categories is projected. Afterwards there is the set of several examples of categories which implement various types of graphs, for example pie, bar, line charts and histograms. Finally there are a few examples of their application mentioned here.

Key words

Widget, Shell, Display, Java, Knihovna, Rámec, Graphical context, Graf, Swing, Standard Widget Toolkit, Abstract Windowing Toolkit, SWTChar, Eclipse.

Prohlášení

Prohlašuji, že jsem tuto diplomovou/bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské/diplomové práce, je legální.

.....

podpis

V Plzni dne 8.6.2017 Martin Voráček

Obsah

Úvod

Předkládaná práce je zaměřena na vytvoření knihovny pro vykreslování grafů v Javě. Za pomocí grafické knihovny „Standard Widget Toolkit“ dále jen SWT.
Data, čísla, různé položky… to vše se obvykle běžně zpracovává v tabulkách. Ale ne vždy můžete z uvedených údajů připravit přehled, stanovit trend vývoje, či jinak správně vyhodnotit danou situaci (například přehled výroby, vývoj prodeje, plánování zakázek, náklady na provoz zařízení…). Pokud však svá data znázorníte graficky, můžete často sledovat nejenom samotné údaje, ale i vazby mezi shromážděnými daty. Najednou se prostě mohou objevit souvislosti, které byste ve vlastních tabulkách snadno přehlédli.
A proč k tomu použít právě knihovnu SWT?

SWT je grafické rozhraní napříč platformami vyvinuté společností IBM. Proč IBM vytvořila další GUI? A nepoužila existující rámce Javy GUI? Chceme-li odpověď na tuto otázku, musíme se vrátit k začátkům Javy. Společnost Sun vytvořila křížovou platformu GUI Abstract Windowing Toolkit dále jen AWT. Rámec AWT používá nativní widgety, ale má problém, způsobuje ztrátu hlavních funkcí platformy. Jinými slovy, pokud má platforma A 1-40 widgetů a Platforma B má 20-25 widgetů, rámcová platforma AWT nabízí pouze průsečík těchto dvou sad. K vyřešení tohoto problému společnost Sun vytvořila nový rámec Swing, který používá emulované widgety namísto nativních widgetů. Tento přístup řešil problém, poskytnutím bohaté sady widgetů, ale způsobil další problémy. Například aplikace ve Swingu již nevypadá jako nativní aplikace. I když má nejnovější vylepšení v JVM, aplikace ve Swingu mají problémy s výkonem, který neexistuje u předešlé AWT. Navíc aplikace ve Swingu spotřebovávají příliš mnoho paměti, což není vhodné pro malé zařízení, jako jsou PDA a mobilní telefony. IBM rozhodla, že žádný z přístupů nesplňuje jejich požadavky. V důsledku toho společnost IBM vytvořila novou knihovnu GUI nazvanou SWT, která řeší problémy, které se vyskytují u rámců AWT a Swing. Rámec SWT využívá nativní widgety prostřednictvím JNI. Není-li widget k dispozici na hostitelské platformě, SWT emuluje nedostupný widget. Pro tyhle výhody je SWT vhodnější, má výhody jak knihovny AWT, tak Swingu. Například i projekt Nebula používá SWT, který sjednocuje řadu widgetů i pro vykreslování grafů. Dále projekt SWTChar, který je také založen na knihovně SWT.

Seznam symbolů a zkratek

AWT ..... Abstract Windowing Toolkit

GUI ..... Grafické uživatelské rozhranní (Graphical User Interface)

IBM..... International Business Machines

JNI ..... Java Native Interface

JVM ..... Java Virtual Machine

SWT..... Standard Widget Toolkit

UML ..... Unified Modeling Language

Cíl práce

Práce předpokládá alespoň základní znalost Javy a není tutoriálem o knihovně SWT. Je o významu a návrhu knihovny pro vykreslování grafů. Tudíž se při popisu tříd a funkcionalit omezím jen na nutné minimum, které se bezprostředně týká návrhu této knihovny.

Knihovnu jsem pojal jako vlastní widget stejně jako v projektu Nebula, který se bude skládat ze dvou hlavní tříd.

`Graf` – tato třída bude obsahovat všechny výpočty a nezbytné procedury.

`DataGrafu-` a tato třída bude jako kontejner pro všechna potřebná data.

První část budu věnovat popisu použitých tříd, aby bylo později jasné jaký význam nebo funkci tam která třída má. Nebudu používat žádné další knihovny. Vše bude jen pomoci knihoven SWT, Math a napsáno ve vývojovém prostředí Eclipse a UML návrh v ArgoUML. Vzhledem k rozsahu této práce se omezím jen na základní vlastnosti a funkce, které by každý graf měl mýt. Slovíčka „widget“ a „plugin“ jsem se rozhodl nepřekládat z důvodů nepříliš vhodných českých překladů.

1 SWT – popis použitých tříd

1.1 Diplay

Tato třída představuje rozhranní mezi SWT a konkrétním operačním systémem, na kterém naši aplikaci spouštíme. Z toho vyplívá, že pro libovolně velké GUI (tvořené i několika okny) budeme vždy potřebovat pouze jedinou instanci této třídy. Za podstatnou funkcí této třídy lze považovat to, že se sama stará o převod událostí, které používáme v naší SWT aplikaci a na události srozumitelné konkrétnímu operačnímu systému. Dále si zde můžeme zmínit, že se tato třída používá k přístupu k informacím o naší platformě a dále automaticky spravuje zdroje, které jsou naší aplikací alokovány. Pro náš účel si vystačíme s jediným konstruktorem:

`Display display = new Display()` - vytváří instanci třídy `Display`.

Z dostupných metod této třídy si zmíníme alespoň ty nejdůležitější:
`display.dispose()` - zcela nezbytná metoda, která slouží k uvolnění zdrojů operačního systému, které jsou drženy naší instancí třídy `Display`. V případě, pokud bychom na tuto metodu ve zdrojovém kódu zapomněli, na většině podporovaných platforem (včetně OS Windows) se naše GUI vůbec nezobrazí. Z těchto důvodů musí být každá vytvořená instance třídy Display "uzavřena" právě touto metodou.
`Widget display.findWidget(int handle)` - vrací specifikovanou komponentu z aktuálně běžící aplikace. V případě, že tato komponenta neexistuje, vrací `null`.
`display.getCurrent()`-vrací běžící instanci třídy `Display`.
`Shell[] display.getShells()` -vrací běžící instance třídy `Shell`, kterou obsahuje aktuální třída `Display`.
`Color display.getSystemColor(int id)`- vrací odpovídající barvu (pro konkrétní platformu), specifikovanou zadanou konstantou, obsaženou v SWT (například `SWT.COLOR_RED`).
`boolean display.readAndDispatch()` - pro základní účely zcela nezbytná metoda. Jejím účelem je kontrolovat události z fronty událostí konkrétního operačního systému a vracet `true`, pokud jsou otevřená nějaká aktivní okna (instance třídy `Shell`).

V opačném případě vrací `false`. Tuto metodu budeme využívat pro kontrolu, zda uživatel kliknul na tlačítko k zavření hlavního okna aplikace. Přestože jsme si zde ukázali pouze několik dostupných metod, je z nich na první pohled patrný charakter této třídy, tj. stát mezi operačním systémem a naší SWT aplikací. Ukázka použití instance této třídy v praxi by neměla větší smysl bez skutečného (viditelného) okna, představovaného třídou `Shell`.

1.2 Shell

Tato třída, představuje hlavní okno našeho GUI a její účel je tedy velmi podobný jako u tříd Frame v AWT, popřípadě JFrame ve Swingu. Z hlediska funkcionality nám tedy poskytuje základní stavební kámen, na který následně umísťujeme další potřebné komponenty (jedná se o instance tříd z balíku org.eclipse.swt.widgets).
Konstruktorů nabízí tato třída hned několik:

`Shell shell = new Shell()` - vytvoří novou instanci třídy `Shell`.

`Shell(Display display)` - umožňuje zadání instance třídy `Display`, kterou bude tato instance využívat.

`Shell(Display display, int style)` - oproti předchozímu konstruktorům umožňuje navíc specifikovat potřebný SWT styl (pro základní specifikaci chování komponenty a jejího vzhledu). Z dostupných konstant si uveďme alespoň `SWT.BORDER`, `SWT.CLOSE`, `SWT.MIN` pro minimalizaci okna, `SWT.MAX` pro maximalizaci okna.

`Shell(int style)` - umožňuje pouze zadání stylu.

`Shell(Shell parent)` - podobně jako můžeme specifikovat `Display`, pro který naši instanci vytváříme, tak nám tento konstruktor umožňuje zadat jinou (rodičovskou) instanci třídy `Shell`.

`Shell(Shell arent, int style)` - tento poslední konstruktor umožňuje oproti předchozímu navíc zadat požadovaný styl. Nyní si uveďme několik metod této třídy:

`Shell[]shell.getShells()` - vrací pole, obsahující všechny potomky našeho

okna.

`boolean shell.isVisible ()` - vrací `true`, je-li okno viditelné. V opačném případě vrací `false`.

`shell.open()` - zcela nepostradatelná metoda. Provádí zobrazení okna tím, že jej dostane do popředí pod danou instancí třídy `Display`. Dále provede několik důležitých akcí, mezi kterými jsou například nastavení viditelnosti na hodnotu `true` okna a přidělení fokusu.

`shell.setMinimumSize(Point size)` - umožňuje specifikovat minimální velikost okna.

`shell.setvisible(boolean visible)` - umožňuje zobrazit okno (`true`), nebo jej skrýt (`false`).

`shell.setText(String nazev)` - nastaví text, který se bude zobrazovat v horní liště nalevo.

`shell.getClientArea()` - vrátí aktuální `Rectagle` což je obdélník, který představuje plochu v našem okně, ve které můžeme kreslit.

1.3 GC Grafický kontext

Grafický systém SWT používá stejnou konvenci souřadnic, která se používá u ovládacích prvků, kde je počátek 0,0 v levém horním rohu a osa x se zvyšuje doprava, zatímco osa y se zvyšuje směrem dolů. Třída Bod se používá k reprezentaci jak polohy (pozice v souřadnicovém systému), tak ofsetu (v SWT není dimenze třídy - velikost obdélníku představuje bod zachycující posun x a y od jeho počátku). Grafika může být nakreslena na všechno, co implementuje org.eclipse.swt.graphics.Drawable. Třída org.eclipse.swt.graphics.GC je grafický kontext, který zapouzdřuje operace, které lze provést a vykreslit. Teď ukážu pár užitečných metod.

`GC.stringExtent(String)` – vrací délku a výšku textu v pixelech.

`GC.setFont(new Font(Display,"Arial",10,1))` – nastavý font kterým se bude vykreslovat text.

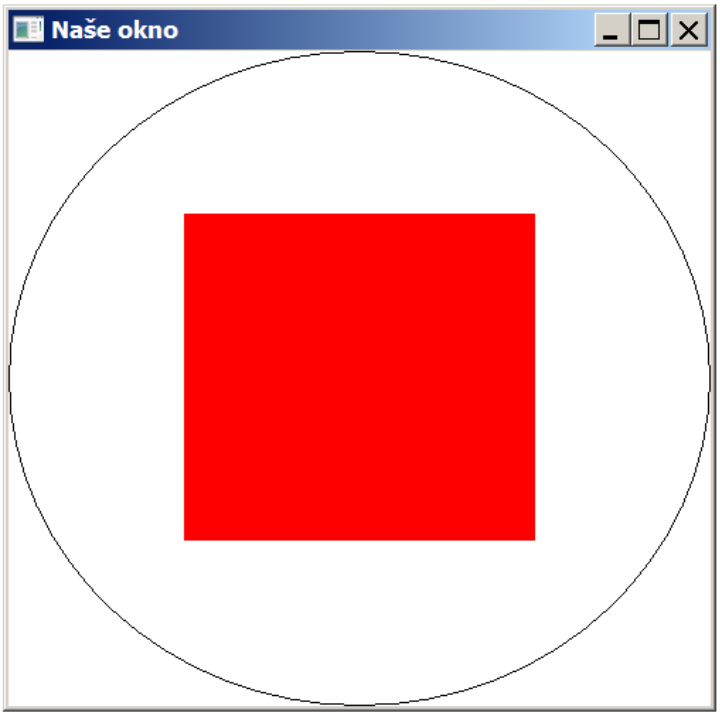
`GC.setBackground(Display.getSystemColor(SWT.COLOR_WHITE))`
Nastaví barvu pozadí.

`GC.setForeground(Display.getSystemColor(SWT.COLOR_BLACK))`
Nastavý barvu textu a podobným věcem.

Uvedu jednoduchý příklad na uvedené třídy:

```
import org.eclipse.swt.SWT…;

public class Main {
public static void main(String[] args){

final Display display = new Display();
final Shell shell = new Shell(display);

shell.setText("Naše okno"); // název okna
   // barva pozadí
shell.setBackground(display.getSystemColor(SWT.COLOR_WHITE));
    //barva písma
shell.setForeground(display.getSystemColor(SWT.COLOR_BLACK));

// Aby bolo vše řádně vykreslováno a překreslováno, musíme
// k oknu přidat potřebnou událost.
shell.addPaintListener(paintEvent ->{GC gc = paintEvent.gc;

// Pro zjištění šířky a výšky použijeme getClientArea().
// gc.drawOval() vykreslí ovál
gc.drawOval(0, 0, shell.getClientArea().width - 1,
shell.getClientArea().height - 1);

// Pro vykreslení vyplněného čtverce musíme změnit barvu
// pozadí grafického kontroloru. Aby byla jiná než okna.
```

```
gc.setBackground(display.getSystemColor(SWT.COLOR_RED));
gc.fillRectangle(shell.getClientArea().width/4,
shell.getClientArea().height/4,shell.getClientArea().width/2,
shell.getClientArea().height/2)
          ;});
shell.open();
       while(!shell.isDisposed())
           if(!display.readAndDispatch()){
                  // pokud bylo zavřeno okno, "uspíme" display
                display.sleep();
                }
       display.dispose();  // nakonec uvolníme držené zdroje
     }
}
```



Obr. 1 Okno vytvořené pomocí SWT

1.4 Widgety

Widgety jsou základními stavebními prvky aplikace GUI. Mysleme na widgety jako

na kostky v legu. V průběhu let se několik widgetů stalo standardem ve všech sadách nástrojů na všech platformách a operačních systémech. Například tlačítko, zaškrtávací políčko nebo textové pole.

1.5 Vlastní widgety

Vlastní widgety se tvoří rozšířením tříd `Composite` nebo `Canvas`. Rozdíl je hlavně v tom, jestli náš nový widget bude už konečný, nebo bude na něj možno umisťovat další widgety. V našem případě si vystačíme s rozšířením třídy `Canvas`.

1.6 Canvas

Přestože jakýkoli ovládací prvek může být nakreslený skrze jeho paintControl podtřída org.eclipse.swt.widgets.Canvas je speciálně navržena pro grafické operace. To lze provést buď pomocí canvas a přidáním paintListener, nebo pomocí podtřídy, čímž vytvoříte znovu použitelné vlastní ovládání. Canvas má řadu stylových bitů, které lze použít k ovlivnění způsobu vykreslování. Proto je vhodný jako základ pro naši knihovnu.

2 Vlastní návrh knihovny

Návrh jsem začal jednoduchým diagramem v jazyce UML.
Následuje založení projektu. Jde o obyčejný projekt, hlavní rozdíl je že v případě knihovny nepotřebujeme metodu:

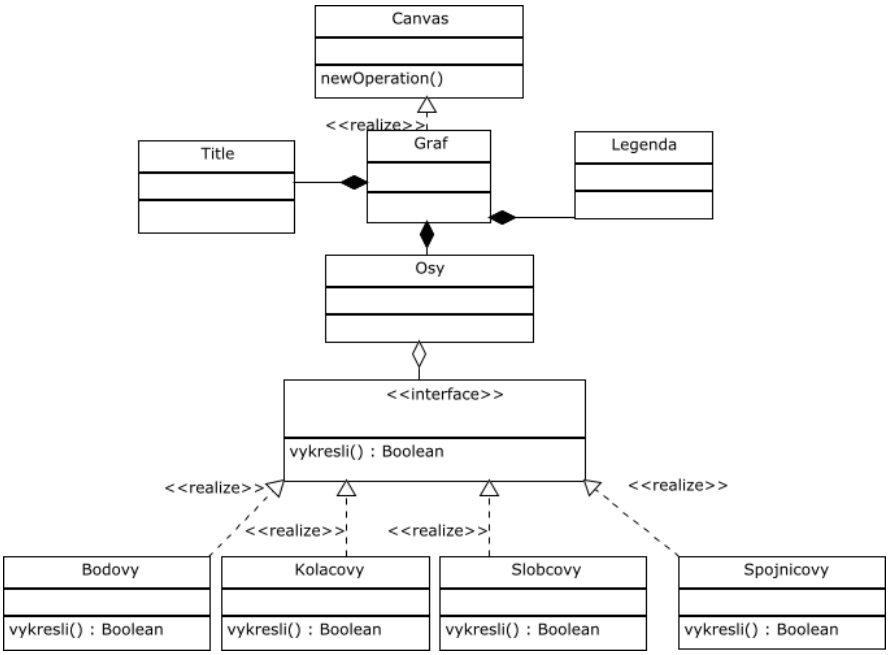
```
public static void main(String[] args) {
…
}.
```

Která nám tvoří vstupní bod do naší aplikace. Protože knihovna je, vždy použita v jiné aplikaci, která už svojí `main ()` třídu má.
Přesto je dobré si ji vytvořit. Velice nápomocná nám bude, při průběžném zkoušení a

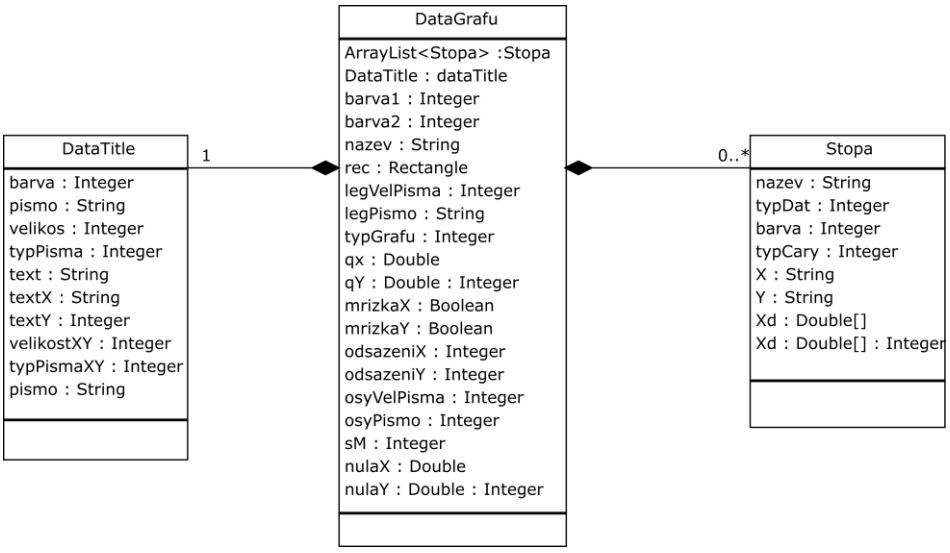
ladění, už napsaného kódu. Při popisu návrhu knihovny budu velmi stručný a tím odkazuji na přílohu.

2.1 Návrh v UML jazyce

V diagramu jsem pro zjednodušení uvedl jen třídy bez atributů. Projekt není velký, celí kód by se dal vtěsnat do několika tříd. Moje rozdělení do následujících tříd. Má jen zpřehlednit a umožnit lepší orientaci.



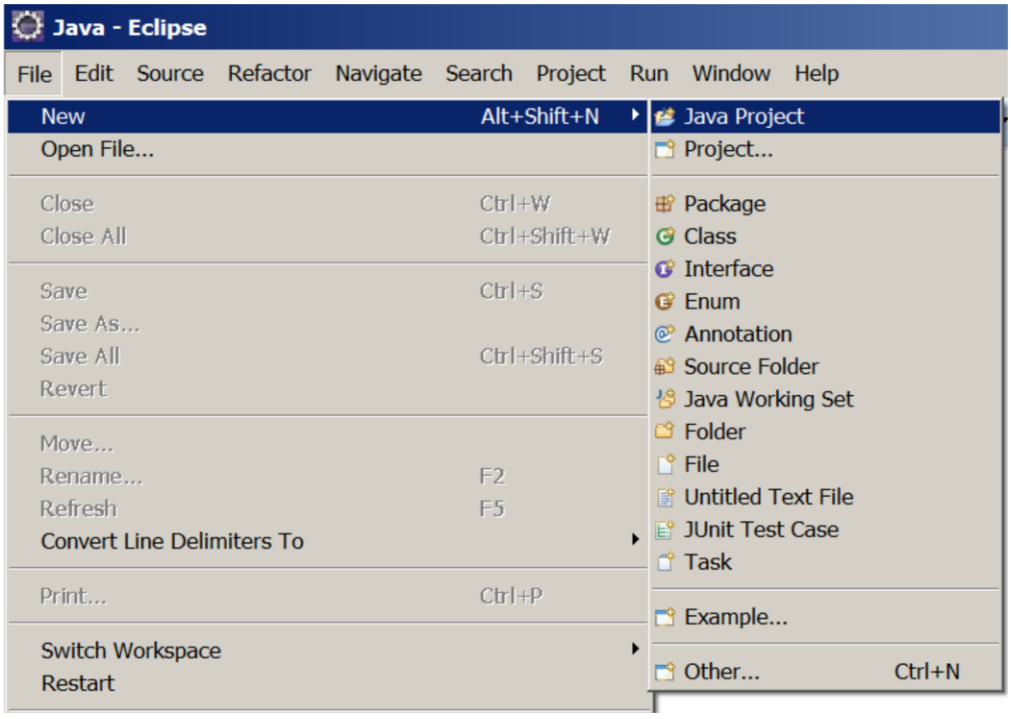
Obr. 2 Návrh třídy Graf bez atributů



Obr. 3 Návrh třídy DataGrafu s atributy

2.1 Založení projektu

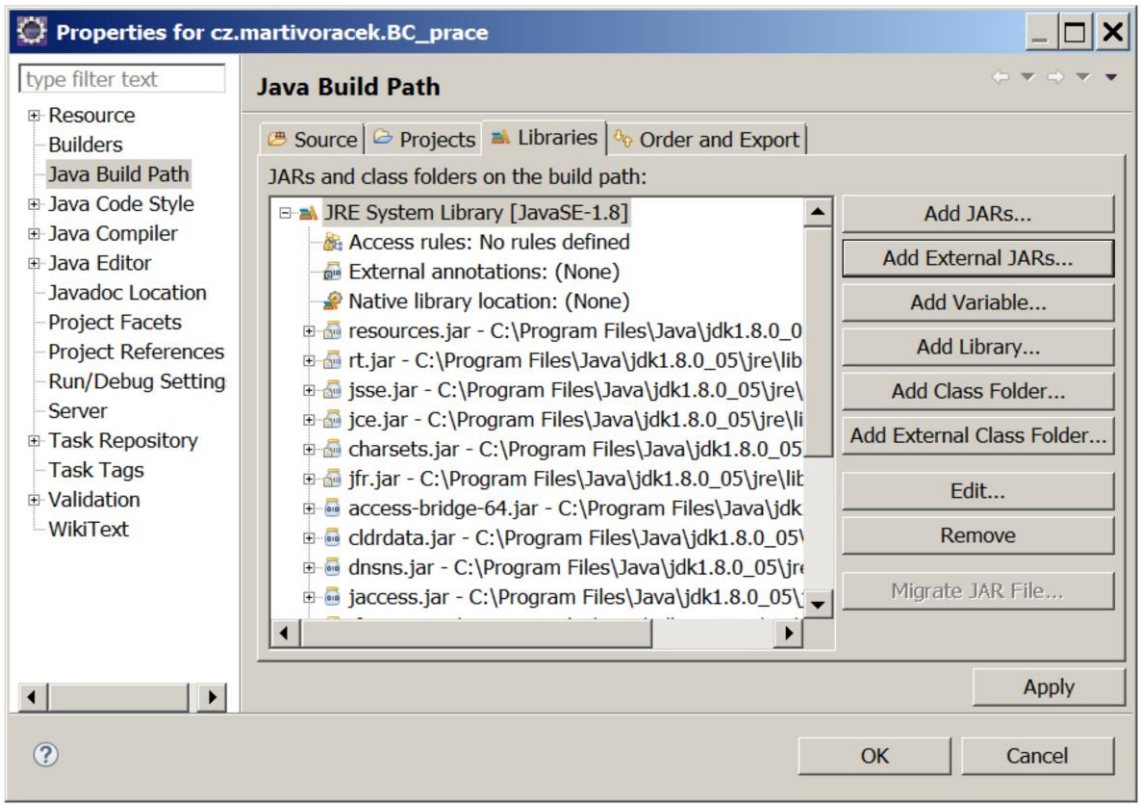
Spustíme vývojové prostředí Eclipse.



Obr. 4 Založení projektu

2.2 Přidání knihovny SWT do projektu

Na stránce www.eclipse.org najdeme a stáhneme zazipovanou verzi SWT knihovny. Zvolte svou platformu a nejvyšší verzi. Stáhněte a rozbalte. Následně přidejte jako externí.



Obr. 5 Přidání knihovny

2.3 Založení tříd

2.3.1 DataGrafu

Do této třídy umístíme všechny proměnné, které se budou týkat grafu. Třída nebude obsahovat žádné procedury ani metody vyjma tzv. (getterů a setterů). Bude poskytovat data pro třídu `Graf`. A zároveň obsahovat veškeré nastavení.

2.3.2 DataTitle

Třída bude mít instanci ve třídě `DataGrafu` a bude mít pod sebou data o nastavení všech popisů. Čímž jsou název grafu a popisy os. Při vykreslování vertikálního popisu bylo třeba otočit souřadný systém os. Pomocí následující metody.

```
Transform tr = new Transform(graf.getDisplay());
          tr.rotate(-90);
          gc.setTransform(tr);
          gc.drawText(this.nazev,-dg.rec.y - dg.rec.height/2
-gc.stringExtent(nazev).x/2,
                    dg.rec.x - gc.stringExtent(nazev).y);
          tr.rotate(90);

          gc.setTransform(tr);
```

2.3.3 Stopa

Třída `DataGrafu` bude mít instanci kolekce `Stopa`. Kde budou informace ke každé stopě.

2.3.4 Graf

Podívejme se na konstruktor `Graf(Shell shell,int,DataGrafu dg).` Jako svůj 3 argument má `DataGrafu`, který před zavolání konstruktoru, musí byt už nastavený. Výpočet je rozdělen do třech tříd.

2.3.5 Title

Stará se o vykreslování názvu grafu a popis os. Zároveň i překreslování při změně velikosti okna.Využívá metodu:
`GC.drawText(String s, int x, int y)`
`s` – řetězec který chcem vykreslit.
`x,y` – souřadnice levého horního rohu textového pole.

2.3.6 Legenda

Vykresluje legendu, která obsahuje názvy stop a barevné značení. A zvětšuje se podle textu nejdelšího názvu.

2.3.7 Osy

Na základě vstupních dat určuje měřítko os. A přizpůsobuje velikosti okna.

2.3.8 TypGrafu – interface

Poskytuje rozhranní pro vykreslování různých typů grafů.

Metodou:

```
public boolean vykresli(GC gc);
```

2.3.9 Koláčový

Na koláčový graf jsme použili metodu.

```
GC.drawArc(int X,int Y,int width,int height,int startAngle,
int endAngle);
```

X, Y – jsou souřadnice levého horního rohu.

Width, height – šířka a výška oválu.

startAngle – startovní úhel.

endAngle – úhel který bude vybarvený (výseč).

Viz příloha E.

2.3.10 Sloupcový

Na sloupce jsme použily metodu.

```
GC.fillRectangle(int X, int Y, int width, int height);
```

X,Y - jsou souřadnice levého horního rohu.

width, height – šířka a výška vyplněného obdélníka. Viz příloha G.

2.3.11 Bodový

Na vykreslení bodů jsem použil.

```
GC.fillOval(int X, int Y, int width, int height);
```

X,Y - jsou souřadnice levého horního rohu.

width, height – šířka a výška vyplněného oválu. Viz příloha A.

2.3.12 Spojnicový

Na spojnicový, mřížku… jsme použili obyčejnou úsečku.

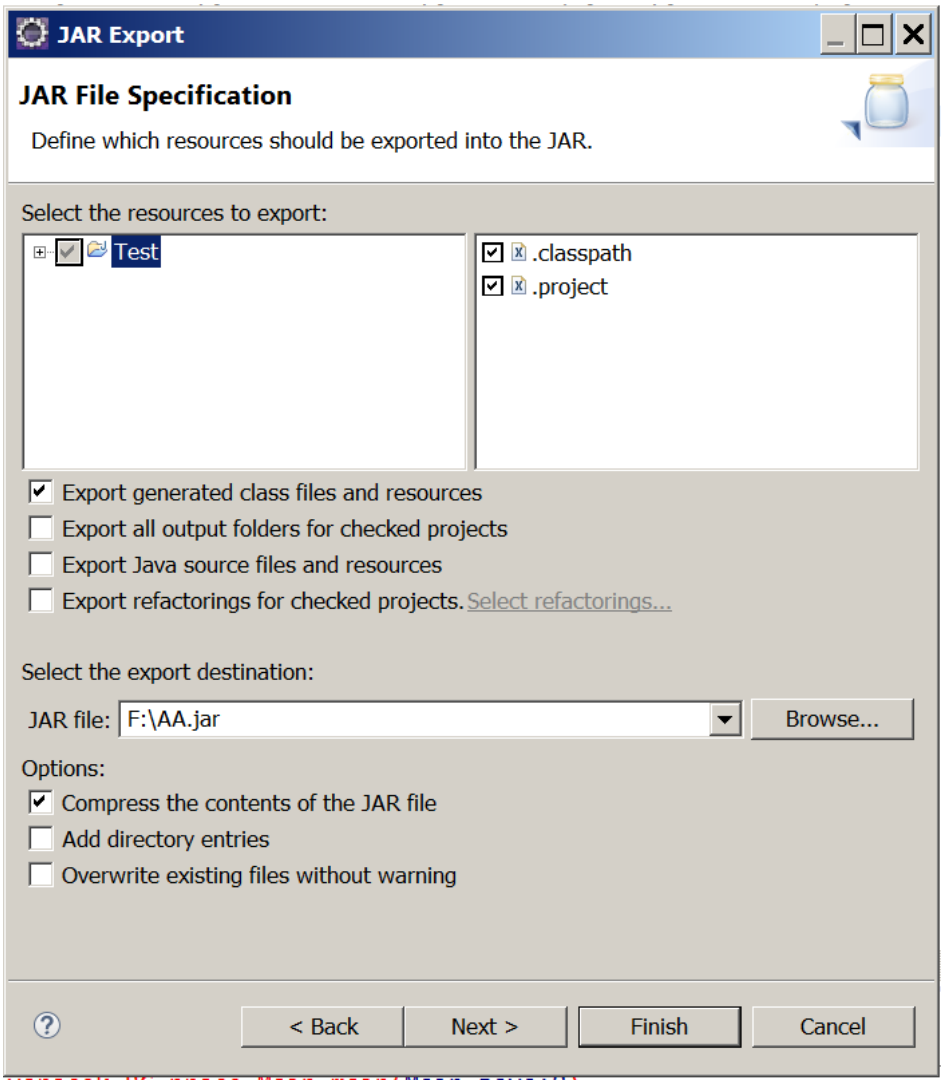
```
GC.drawLine(int x1, int y1, int x2, int y2);
```

x1,y1 – souřadnice počátečního bodu

x2,y2 – souřadnice koncového bodu. Viz příloha I.

2.4 Exportování

Pravým tlačítkem na projekt a dáme export.



Obr. 6 Export do .jar souboru

Vybereme náš projekt, umístění a dáme finish.

3 Ukázky funkce naší knihovny

V téhle kapitole si konečně vyzkoušíme, co se nám povedlo vytvořit.
Na začátek si vytvoříme takovou šablonu, abyste nemuseli stále opisovat celí kód.

3.1 Vytvoření nového projektu

Nový projekt vytvoříme stejně jako v podkapitole 2.2 Založení projektu. Jen nám tentokrát bude stačit jenom jedna třída s metodou `main()`.

```
import org.eclipse.swt.SWT…;
public class Main {
public static void main(String[] args){
final Display display = new Display();
final Shell shell = new Shell(display);
//
// Tady se bude nacházet kód při našich ukázkách
//
shell.open();
      while(!shell.isDisposed())
          if(!display.readAndDispatch()){
                 // pokud bylo zavřeno okno, "uspíme" display
               display.sleep();
               }
      display.dispose();  // nakonec uvolníme držené zdroje
     }
}
```

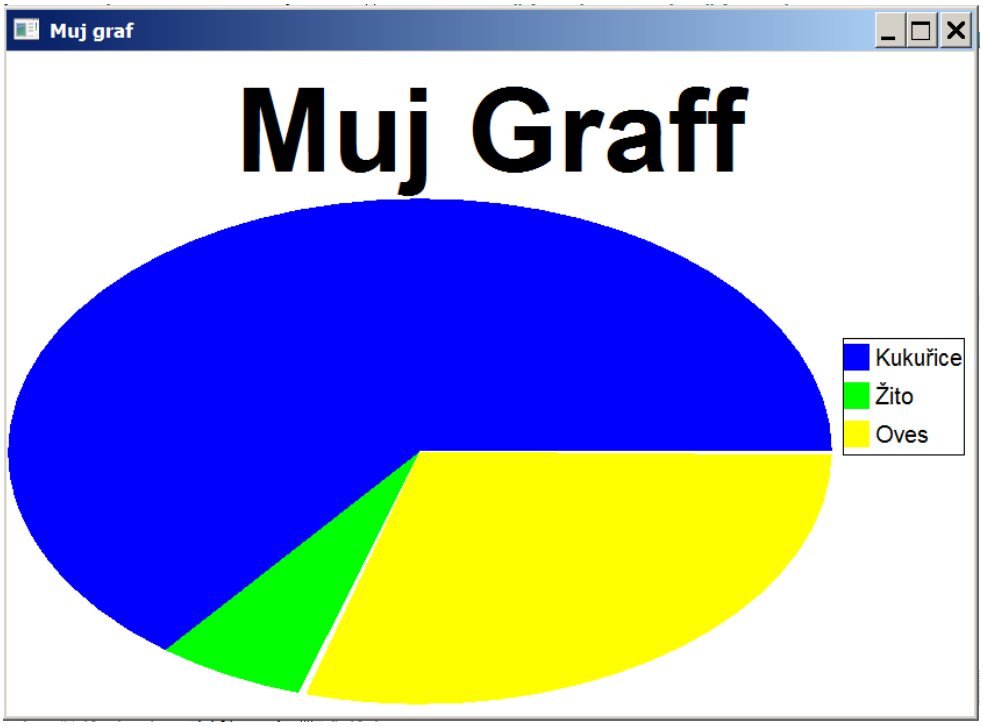
3.1.1 Přidání naší knihovny

Přidání naší knihovny bude stejné, jako bylo s knihovnou SWT.

Jenom si najdeme, kam jsme si naši knihovnu uložili. Následující ukázky jsou upravovány bez přímého zásahu do knihovny.

3.2 Ukázka první graf koláčový

```
*
*
DataGrafu dg = new DataGrafu("Nadpis");
double[] i =  {110.0};
double[] i2 =  {10.0};
double[] i3 = {50.0};
dg.typGrafu = 1;

Stopa s = new Stopa("Kukuřice",i);
Stopa s2 =  new Stopa("Žito",i2);
Stopa s3 = new Stopa("Oves",i3);
s.setBarva(SWT.COLOR_BLUE);
dg.stopa.add(s);
s2.setBarva(SWT.COLOR_GREEN);
dg.stopa.add(s2);
s3.setBarva(SWT.COLOR_YELLOW);
dg.stopa.add(s3);
dg.title.setTextX("Osa X");
dg.title.setTextY("OsaY");
dg.title.setText("Muj Graff");
dg.nazev = "Muj graf";
dg.barva1 = SWT.COLOR_WHITE;
Graf graf = new Graf(shell,SWT.DOUBLE_BUFFERED,dg);
*
*
```

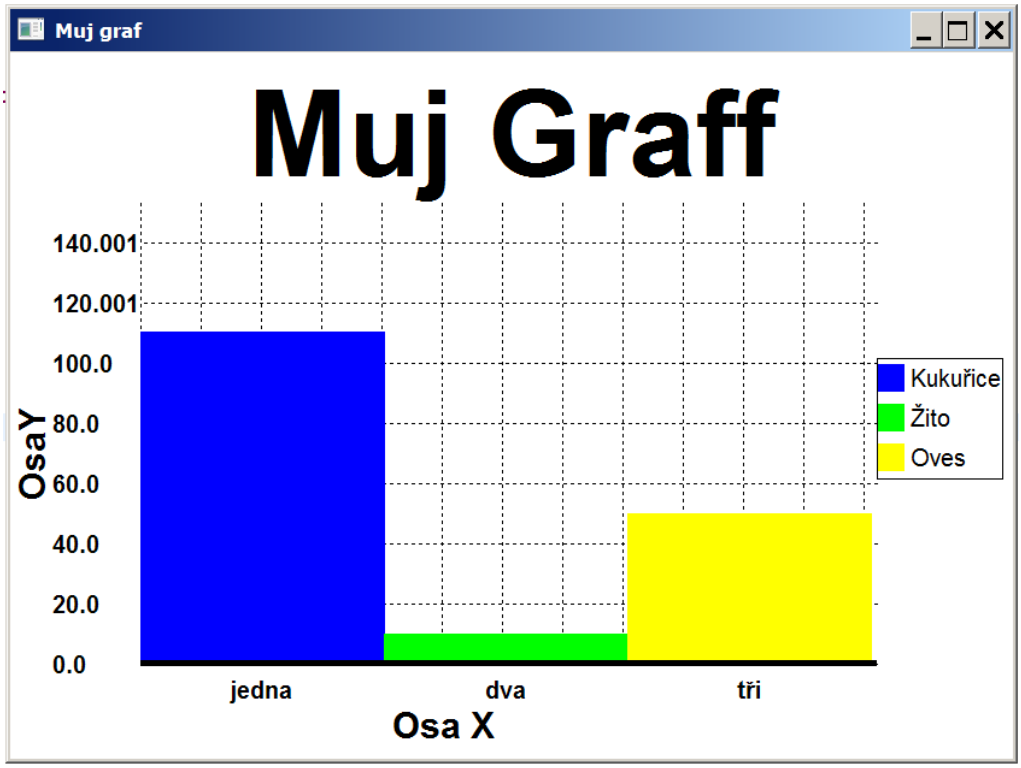


Obr. 7 Ukázka koláčového grafu

3.3 Ukázka druhá graf sloupcový

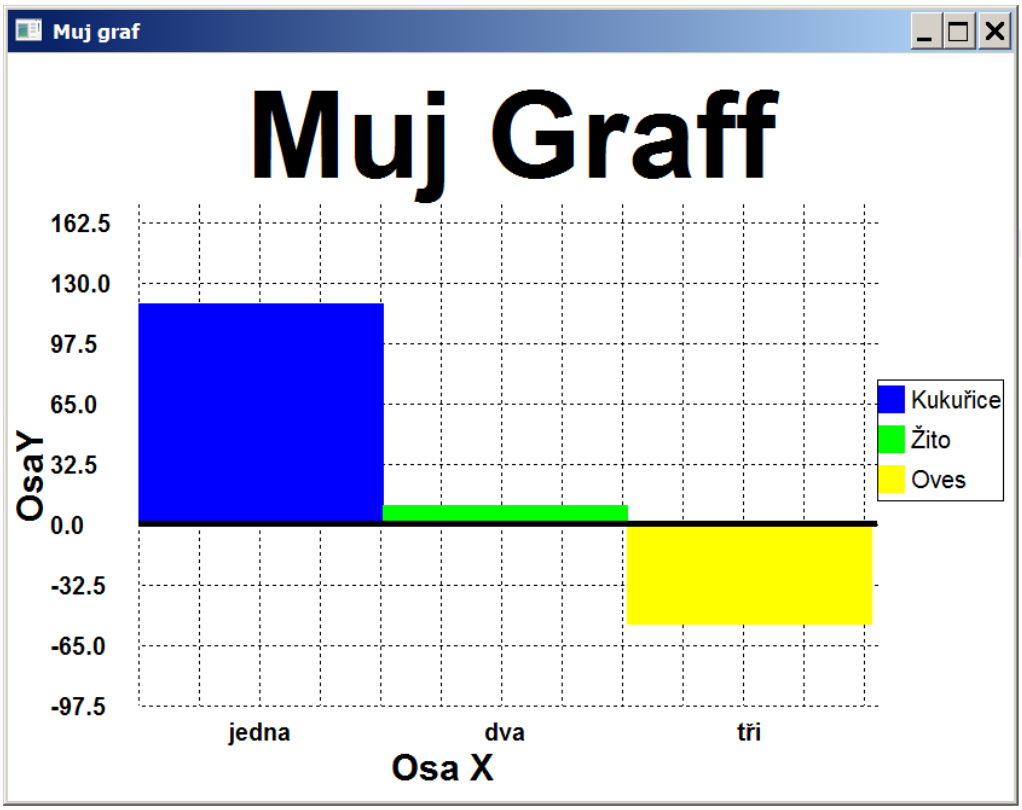
```
*
*
DataGrafu dg = new DataGrafu("Nadpis");
double[] i =  {110.0};
double[] i2 =  {10.0};
double[] i3 = {50.0};
dg.typGrafu = 2; //typ grafu soubcový
Stopa s = new Stopa("Kukuřice",i,"jedna");
Stopa s2 =  new Stopa("Žito",i2,"dva");
Stopa s3 = new Stopa("Oves",i3,"tři");
s.setBarva(SWT.COLOR_BLUE);
dg.stopa.add(s);
s2.setBarva(SWT.COLOR_GREEN);
dg.stopa.add(s2);
s3.setBarva(SWT.COLOR_YELLOW);
dg.stopa.add(s3);
```

```
dg.title.setTextX("Osa X");
dg.title.setTextY("OsaY");
dg.title.setText("Muj Graff");
dg.nazev = "Muj graf";
dg.barva1 = SWT.COLOR_WHITE;
Graf graf = new Graf(shell,SWT.DOUBLE_BUFFERED,dg);
*
*
```



Obr. 8 Ukázka sloupcového grafu

Malá úprava: `double[] i3 = {-50.0};`



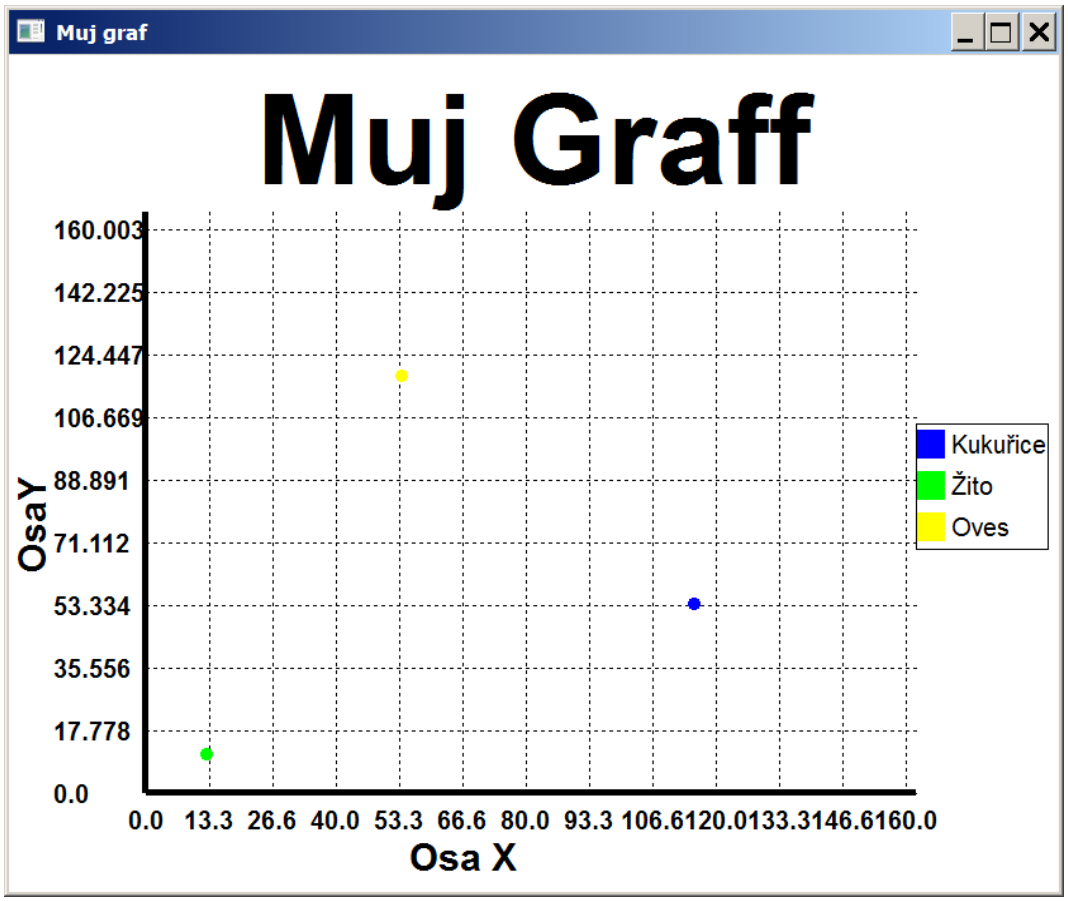
Obr. 9 Ukázka se zápornou hodnotou

3.4 Ukázky třetí graf bodový

Bere vždy jen první hodnotu v poli.

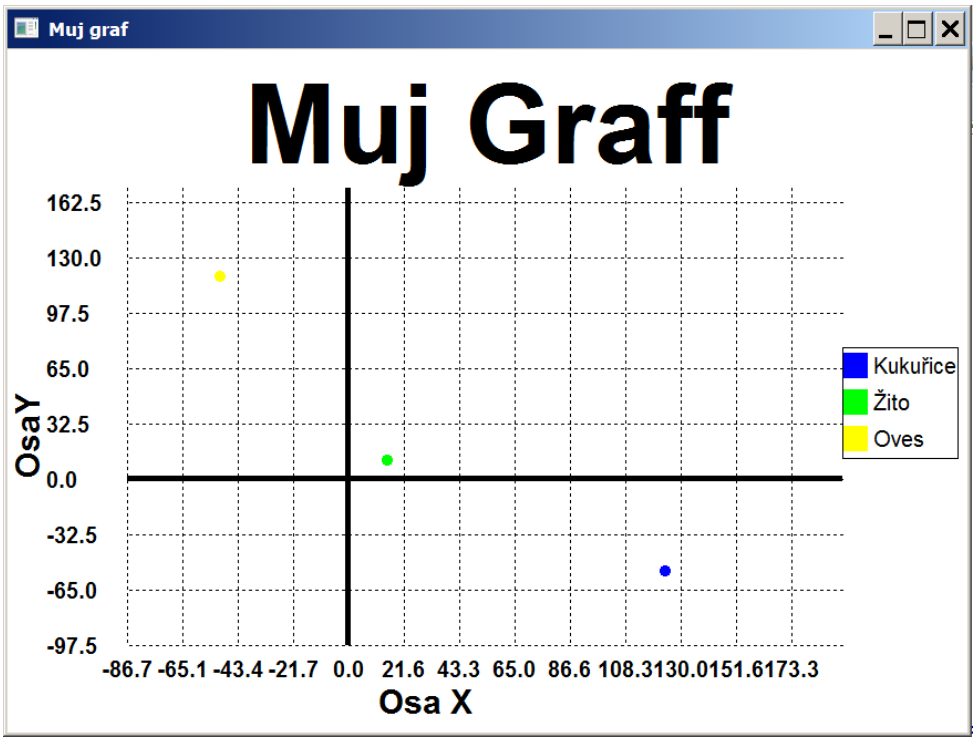
```
DataGrafu dg = new DataGrafu("Nadpis");
double[] i =  {110.0,150,100,122,90};
double[] i2 =  {10.0,56,45,60,45};
double[] i3 = {50.0,45,45,65,30};
dg.typGrafu = 3;
Stopa s = new Stopa("Kukuřice",i,i3);
Stopa s2 =  new Stopa("Žito",i2,i2);
Stopa s3 = new Stopa("Oves",i3,i);
s.setBarva(SWT.COLOR_BLUE);
dg.stopa.add(s);
s2.setBarva(SWT.COLOR_GREEN);
dg.stopa.add(s2);
s3.setBarva(SWT.COLOR_YELLOW);
```

```
dg.stopa.add(s3);
dg.title.setTextX("Osa X");
dg.title.setTextY("OsaY");
dg.title.setText("Muj Graff");
dg.nazev = "Muj graf";
dg.barva1 = SWT.COLOR_WHITE;
Graf graf = new Graf(shell,SWT.DOUBLE_BUFFERED,dg);
```



Obr. 10 Ukázka bodový graf, jen kladné hodnoty

Malá úprava: `double[] i3 = {-50.0,45,-45,-65,-30};`



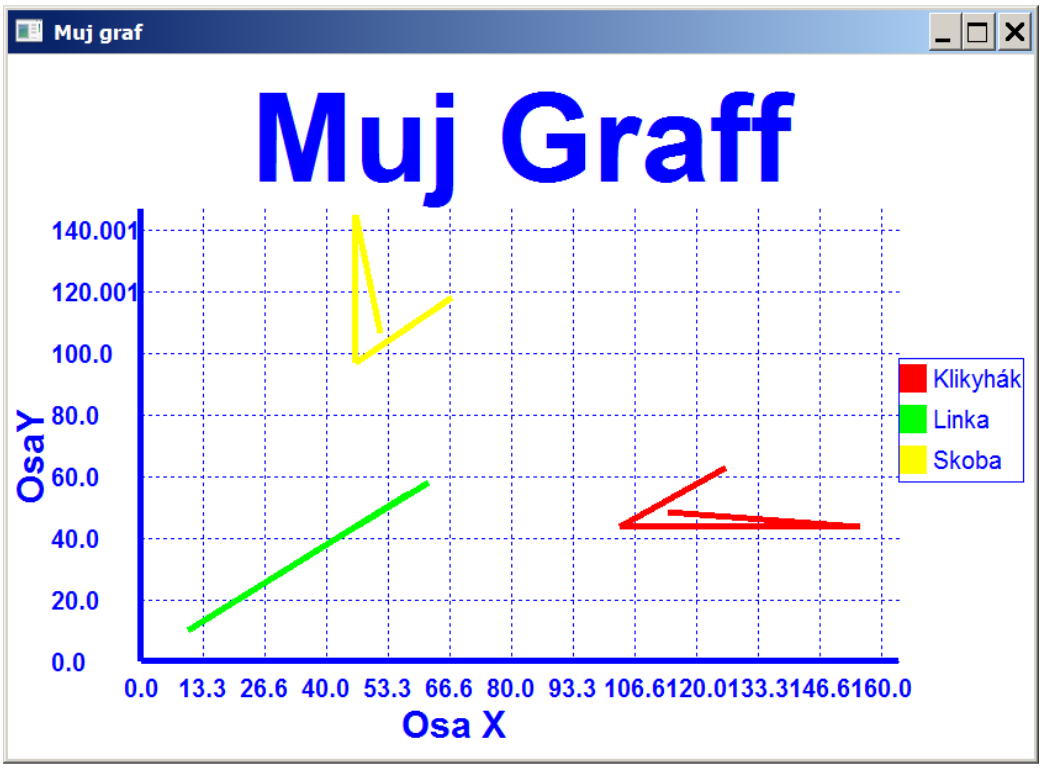
Obr. 11 Při vložení záporné hodnoty

3.5 Ukázka čtvrtá graf spojnicový

Tento typ už využívá celé pole hodnot. Dále, jsem změnil barvu textu. A třetí stopy.

```
DataGrafu dg = new DataGrafu("Nadpis");
double[] i =  {110.0,150,100,122,90};
double[] i2 =  {10.0,56,45,60,45};
double[] i3 = {50.0,45,45,65,30};
dg.typGrafu = 4;
dg.title.setBarva(SWT.COLOR_BLUE);
Stopa s = new Stopa("Klikyhák",i,i3);
Stopa s2 =  new Stopa("Linka",i2,i2);
Stopa s3 = new Stopa("Skoba",i3,i);
s.setBarva(SWT.COLOR_RED);
dg.stopa.add(s);
s2.setBarva(SWT.COLOR_GREEN);
dg.stopa.add(s2);
s3.setBarva(SWT.COLOR_YELLOW);
```

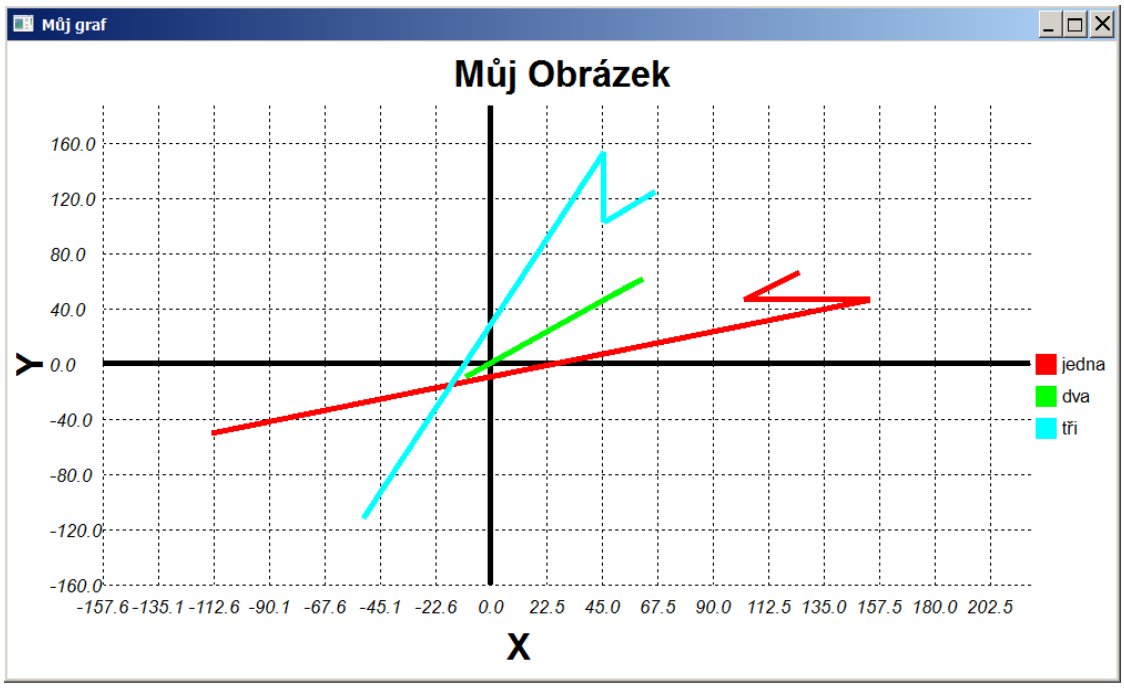
```
dg.stopa.add(s3);
dg.title.setTextX("Osa X");
dg.title.setTextY("OsaY");
dg.title.setText("Muj Graff");
dg.nazev = "Muj graf";
dg.barva1 = SWT.COLOR_WHITE;
Graf graf = new Graf(shell,SWT.DOUBLE_BUFFERED,dg);
```



Obr. 12 Ukázka spojnicového grafu s kladnými hodnotami

Po několika úpravách včetně této:

```
double[] i =  {-110.0,150,100,122,90};
double[] i2 =  {-10.0,56,45,60,45};
double[] i3 = {-50.0,45,45,65,30};
```



Obr. 13 Ukázka spojnicového grafu se zápornými hodnotami

Závěr

Práce splnila, moje očekávání. Při své jednoduchosti grafy vypadají docela zdařile.

Použití SWT knihovny je poměrně snadné, ale naprogramovat úplnou knihovnu se všemi náležitostmi, by bylo velice pracné.

Vstup různých kombinací dat jsem vyřešil ve třídě Stopa. Několika konstruktory. A následném znemožnění použití nevhodných dat, na konkrétní typ grafu.

Tak znemožnit vnesení chyby do vykreslování. Graf je navržený tak, aby se měnil s velikostí okna, ale následně vytváří odchylky od vstupních dat. V průběhu návrhu jsem se asi dopustil nějaké chyby. Knihovna bude mít asi, i jiné chyby na ty, jsem ale nepřišel.

Nejedná se v praxi o příliš použitelnou knihovnu, ale spíš o ukázku jak by to mohlo vypadat.

Seznam literatury a informačních zdrojů

[1] Winchester, J. Graphics Context - Quick on the draw. Eclipse [online]. Leden 2003. 7 (1). [cit. 2017-06-08]. Dostupné z < http://www.eclipse.org/articles/Article-SWT graphics/SWT_graphics.html >.

[2] java2s.com. SWT Tutorial [online]. červen 2017.. Dostupné z <http://www.java2s.com/Tutorial/Java/0280__SWT/Catalog0280__SWT.htm>.

Příloha A třída Bodovy

```
package cz.martinvoracek.bp;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.GC;

public class Bodovy implements TypGrafu{
    private Graf graf;
    private DataGrafu dg;

      Bodovy(Graf graf,DataGrafu dg){
            this.dg = dg;
            this.graf = graf;
            dg.mrizkaX = true; // v případě true vykreslí mřížku
            dg.mrizkaY =true;
      }

      public boolean vykresli(GC gc) {

            if(dg.stopa.get(0).getTypDat()!=3)
                  return false; // v případě chybných dat se ukončí
            /// vykreslení mřížky
            if(dg.mrizkaY){
            gc.setLineStyle(SWT.LINE_DOT);
            for(int i = 0; i*dg.sM< dg.rec.width;i++){
                  gc.drawLine(i*dg.sM + dg.rec.x, dg.rec.y, i*dg.sM +
dg.rec.x, dg.rec.y+dg.rec.height);

      }
            }
            if(dg.mrizkaX){
            for(int i = 0; i*dg.sM < dg.rec.height;i++){
                  gc.drawLine(dg.rec.width+dg.rec.x,dg.rec.height+
dg.rec.y-(i*dg.sM),dg.rec.x,
                              dg.rec.height+dg.rec.y- (i*dg.sM));

      }
            gc.setLineStyle(SWT.LINE_SOLID);
            gc.setLineWidth(5);
            gc.drawLine(dg.rec.x,(int)(dg.rec.y+dg.rec.height-dg.nulaY),
dg.rec.width+dg.rec.x,
                        (int)(dg.rec.y+dg.rec.height-dg.nulaY));
            gc.drawLine((int)(dg.rec.x+dg.nulaX),
dg.rec.y+dg.rec.height,(int)(dg.rec.x +dg.nulaX), dg.rec.y);
            gc.setLineWidth(1);
            }
            for(Stopa s:dg.stopa){

      gc.setBackground(graf.getDisplay().getSystemColor(s.getBarva()));
            gc.fillOval((int)(dg.rec.x+dg.nulaX
+(Math.floor(s.Xd[0]/dg.qx))+5),
                        (int)(dg.rec.y+dg.rec.height-dg.nulaY-
(s.Yd[0]/dg.qy))-5, 10, 10);
            }

      return true;
      }
}
```

Příloha B třída DataGrafu

```
package cz.martinvoracek.bp;

import java.util.ArrayList;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Rectangle;

public class DataGrafu {

      public ArrayList<Stopa> stopa = new ArrayList<Stopa>(); //Kolekce
Stop
    public DataTitle title;   // nazev grafu

    public int barva1,barva2; //barva popředí a pozadí grafu
    public String nazev; //název okna grafu
    public Rectangle rec; // aktuální obdélník
    // Nastavení Legendy ////////////////////////////////////////////
    public int legVelPisma; // velikst písma legendy
    public String legPismo; // typPisma legendy
    ////////////////////////////////////////////////////////////////
    public int typGrafu; // typGrafu koláčový, slobcový, bodový ..

    public double qy; // Kvocient přepčtu z čísel na pixeli
    public double qx;

    public boolean mrizkaX;
    public boolean mrizkaY;

    public int odsazeniX;
    public int odsazeniY;

    // nastavení osy
    public int osyVelPisma;
    public int osyPismo;

    public final int sM = 50; // konstanta mřížky

    public double nulaX;
    public double nulaY;

    public DataGrafu(String s){
      title = new DataTitle(s);
      this.nazev = s;
      this.barva1 = SWT.COLOR_BLUE;
      this.legPismo = "Arial";
      this.legVelPisma = 10;
      this.mrizkaX = false;
      this.mrizkaY = false;
      this.typGrafu = 2;
      this.odsazeniX = 0;
      this.odsazeniY = 0;
    }
}
```

Příloha C třída DataTitle

```
package cz.martinvoracek.bp;

import org.eclipse.swt.SWT;

public class DataTitle {

	////// Hlavní title
	private int barva = SWT.COLOR_BLACK;
	private String pismo;
	private int velikos;
	private int typPisma;
	private String text;

	////// Title os grafu
	private String textX;
	private String textY;
	private int velikosXY;
	private int typPismaXY;
	private String pismoXY;

	// Konstruktor s výchozímy hodnotamy
	DataTitle(String s){
		this.text = s;
		this.pismo = "Arial";
		this.velikos = 50;
		this.typPisma = 1;
		this.textX = s;
		this.textY = s;
		this.pismoXY = "Arial";
		this.velikosXY = 15;
		this.typPismaXY = 1;
		this.barva = SWT.COLOR_BLACK;
	}

	public int getBarva() {return barva;}
	public String getTextX() {return textX;}
	public void setTextX(String textX) {this.textX = textX;}
	public String getTextY() {return textY;}
	public void setTextY(String textY) {this.textY = textY;}
	public int getVelikosXY() {return velikosXY;}
	public void setVelikosXY(int velikosXY) {this.velikosXY =
velikosXY;}
	public int getTypXY() {return typPismaXY;}
	public void setTypXY(int typXY) {this.typPismaXY = typXY;}
	public String getPismoXY() {return pismoXY;}
	public void setPismoXY(String pismoXY) {this.pismoXY = pismoXY;}
	public void setBarva(int barva) {this.barva = barva;}
	public void setPismo(String pismo) {this.pismo = pismo;}
	public void setVelikos(int velikos) {this.velikos = velikos;}
	public void setTyp(int typ) {this.typPisma = typ;}
	public void setText(String text) {this.text = text;}
	public String getPismo() {return pismo;}
	public int getVelikos() {return velikos;}
	public int getTyp() {return typPisma;}
	public String getText() {return this.text;}
}
```

Příloha D třída Graf

```
package cz.martinvoracek.bp;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.GC;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Canvas;
import org.eclipse.swt.widgets.Composite;



public class Graf extends Canvas{
	private DataGrafu dg;
	private Title title;
	private Legenda legenda;
	private Osy osy;

	public  Graf(Composite arg0, int arg1,DataGrafu dg) {
		super(arg0, arg1);
		this.dg = dg;
		this.dg.rec = this.getClientArea();
		title = new Title(this,dg);
		legenda = new Legenda(this,dg);
		osy = new Osy(this,dg);
		this.getShell().setText(dg.nazev);
		//Výchozí nastavení barvy pozadí

this.setBackground(this.getDisplay().getSystemColor(SWT.COLOR_WHITE));

this.setForeground(this.getDisplay().getSystemColor(SWT.COLOR_BLACK));

	 // Přidání metodu která se nám bude starat o vykreslování scény
		this.addPaintListener(paintEvent -> {GC gc = paintEvent.gc;
		vykresli(gc);
		});

	// výchozí nastavení okna
	this.getShell().setLayout(new FillLayout());
	this.getShell().setMinimumSize(800, 615);

	}

	private void vykresli(GC gc) {
	    dg.rec = this.getClientArea();
	    title.vykresli(gc);
	    legenda.vykresli(gc);
	    osy.vykresli(gc);
	}
}
```

Příloha E třída Kolacovy

```
package cz.martinvoracek.bp;
import org.eclipse.swt.graphics.GC;


public class Kolacovy implements TypGrafu{

      private Graf graf;
    private DataGrafu dg;

      Kolacovy(Graf graf, DataGrafu dg) {
            this.dg = dg;
            this.graf = graf;
            dg.mrizkaX = false;
            dg.mrizkaY = false;
      }

      public boolean vykresli(GC gc){
            if(dg.stopa.get(0).getTypDat()!=2)
                  return false;

            double d=0;
            double d1=0;
            double d2;

            for(Stopa s:dg.stopa){
            d = d + Math.abs(s.Xd[0]);
            }
            for(Stopa s :dg.stopa){
            d2 = ((360/d)*Math.abs(s.Xd[0]));

      gc.setBackground(this.graf.getDisplay().getSystemColor(s.getBarva()
));
            gc.fillArc(dg.rec.x+1,dg.rec.y+1,dg.rec.width-
10,dg.rec.height-1,(int)d1,(int)d2);
            d1 =d1+d2;
            }
      return true;
      }
}
```

Příloha F třída Osy

```
package cz.martinvoracek.bp;

import java.util.ArrayList;
import org.eclipse.swt.graphics.Font;
import org.eclipse.swt.graphics.GC;


public class Osy {

      private DataGrafu dg;
      private Title title;
      private Graf graf;
      private TypGrafu tg;
```

```
	private Font fontP;
	private double maxY;
	private double minY;
	private double maxX;
	private double minX;
	private final int sM = 50;
	private int pocCtv;
	private double pocetZap;
	Osy(Graf graf,DataGrafu dg){
		this.dg = dg;
		this.graf = graf;
		this.title = new Title(this.graf,dg);
		this.nejKladna(dg.stopa);
		this.nejZapor(dg.stopa);

		this.fontP = new Font(graf.getDisplay(),"Arial",10,2);

		dg.odsazeniX = 0;
		dg.odsazeniY =0;

	}
	public void vykresli(GC gc){
		dg.qy = (this.maxY + Math.abs(this.minY))/(dg.rec.height-sM);
		dg.qx = (this.maxX + Math.abs(this.minX))/(dg.rec.width-
sM*2);
		String ss;
		if(dg.typGrafu!=1){/////////
			title.vykresliX(gc);
			title.vykresliY(gc);

		/// Vykreslení číselné stupnice Y
		for(int i = 0;i*sM < dg.rec.height;i++){
			this.pocCtv = i;
		}
		    dg.odsazeniX =
gc.stringExtent(Double.toString(Math.floor(cy(0,pocCtv)*100+0.5)/100)).y;
		    dg.rec.height -= dg.odsazeniX;
		for(int i = 0; i*sM < dg.rec.height;i++){
		    gc.setFont(fontP);
		      ss =
Double.toString(Math.floor(cy(i,pocCtv)*100+0.5)/100);
		      gc.drawText(ss, dg.rec.x, dg.rec.height+ dg.rec.y-
(i*sM)-gc.textExtent(ss).y/2);
		      if(dg.odsazeniY < gc.stringExtent(ss).x){
		      dg.odsazeniY = gc.stringExtent(ss).x;}
		}
		dg.rec.x += dg.odsazeniY;
		dg.rec.width -= dg.odsazeniY;
		if(dg.typGrafu != 2){

		      for(int i = 0; i*sM < dg.rec.width;i++){//zjistíme
aktualní pocet ctvercu v osy X
		            this.pocCtv = i;
		      }

		     for(int i = 0; i*sM < dg.rec.width;i++){
		       gc.setFont(fontP);
		       ss = Double.toString(Math.floor(cx(i,pocCtv)*10)/10);
		       gc.drawText(ss, dg.rec.x+(i*sM)-gc.textExtent(ss).x/2,
dg.rec.height+dg.rec.y+10);
		     }
```

```
		}
		}
		vyber(gc);
	}

	private void vyber(GC gc){

		switch(dg.typGrafu){
		case 1:
			tg= new Kolacovy(this.graf,this.dg);
			tg.vykresli(gc);
			break;
		case 2:
			tg = new Sloubcovy(this.graf,this.dg);
			tg.vykresli(gc);
			break;
		case 3:
			tg = new Bodovy(this.graf,this.dg);
			tg.vykresli(gc);
			break;
		case 4:
			tg = new Spojnicovy(this.graf,this.dg);
			tg.vykresli(gc);
			break;
		}
	}

	private void nejKladna(ArrayList<Stopa> p){
		double dY= 0;
		double dX = 0;
		for(Stopa s:p){
			for(int i = 0;i<s.Xd.length;i++){
			if(s.Xd[i]>dY)
				dY = s.Xd[i]+sM;
		}
			if(s.Yd!=null){
			for(int i = 0;i<s.Yd.length;i++){
				if(s.Yd[i]>dX)
					dX = s.Yd[i]+sM;
			}
			}
		}
		this.maxY = dY;
		this.maxX = dY;
	}
	private void nejZapor(ArrayList<Stopa> p){
		double dY = 0;
		double dX = 0;
		for(Stopa s:p){
			for(int i = 0;i<s.Xd.length;i++){
			if(s.Xd[i]<dY)
				dY = s.Xd[i]-sM;
		}
			if(s.Yd!=null){
			for(int i = 0;i<s.Yd.length;i++){
				if(s.Yd[i]<dX)
					dX = s.Yd[i]-sM;
			}
			}
		}
		this.minY = dY;
```

```
            this.minX = dX;
      }

      private double  cy(int i,int j){
            double p1 = 0;

            p1 = (this.maxY + Math.abs(this.minY))/j;

            pocetZap = Math.floor((Math.abs(this.minY)/p1));
          dg.nulaY =sM*pocetZap;

            return generator(p1)*(i-pocetZap);
      }
      private double cx(int i, int j){
            double p2 = 0;
            p2 = (this.maxX + Math.abs(this.minX))/j;
            pocetZap = (int)Math.floor((Math.abs(this.minX)/p2));
            dg.nulaX = sM*pocetZap;
            System.out.println(dg.nulaY);
            return generator(p2)*(i-pocetZap);
      }

      private double generator(double d){
            double d2 = 0.0001000;
            while(d>d2){
                  d2 = d2+0.001;
                  while(d>d2){
                        d2 = d2+0.0005;
                  }
            }
            return d2;
      }
}
```

Příloha G třída Sloupcový

```
package cz.martinvoracek.bp;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.GC;



public class Sloubcovy implements TypGrafu{


      //public int velPismaX;
      private Graf graf;
      private DataGrafu dg;


      Sloubcovy(Graf graf, DataGrafu dg) {


            this.graf = graf;
            this.dg = dg;
            dg.mrizkaX = true;
            dg.mrizkaY = true;
```

```
	}

	public boolean vykresli(GC gc) {

		if(dg.stopa.get(0).getTypDat()!=1)
			 return false;
			/// vykreslení mřížky
			if(dg.mrizkaY){
			gc.setLineStyle(SWT.LINE_DOT);
			for(int i = 0; i*dg.sM< dg.rec.width;i++){
				gc.drawLine(i*dg.sM + dg.rec.x, dg.rec.y, i*dg.sM
+ dg.rec.x,
						dg.rec.y+dg.rec.height);


		}
			}
			if(dg.mrizkaX){
			for(int i = 0; i*dg.sM < dg.rec.height;i++){
				gc.drawLine(dg.rec.width+dg.rec.x,dg.rec.height+
dg.rec.y-(i*dg.sM),
						dg.rec.x,dg.rec.height+dg.rec.y-
(i*dg.sM));

		}


			int i = dg.rec.width/dg.stopa.size();
			int c=0;
			System.out.println(dg.nulaY);
			for(Stopa s:dg.stopa){

				if(s.getTypDat()!= 3){//nevykreslujese

	gc.setBackground(graf.getDisplay().getSystemColor(s.getBarva()));

	gc.fillRectangle(dg.rec.x+c,(int)(dg.rec.y+dg.rec.height-dg.nulaY),
						i, -(int)(s.Xd[0]/dg.qy));

	gc.setBackground(graf.getDisplay().getSystemColor(dg.barva1));
				gc.drawText(s.Y,dg.rec.x+c+ (i/2)-
(gc.textExtent(s.Y).x/2),
						dg.rec.y+dg.rec.height+10);
				c = c+i-1;
					}
			}
			gc.setLineStyle(SWT.LINE_SOLID);
			gc.setLineWidth(5);
			gc.drawLine(dg.rec.x,(int)(dg.rec.y+dg.rec.height-
dg.nulaY), dg.rec.width+dg.rec.x,
				(int)( dg.rec.y+dg.rec.height-dg.nulaY));
			gc.setLineWidth(1);
			}
	return true;
	}
}
```

Příloha I třída Spojnicovy

```
package cz.martinvoracek.bp;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.GC;

public class Spojnicovy implements TypGrafu {

	private Graf graf;
	private DataGrafu dg;

	Spojnicovy(Graf graf,DataGrafu dg){
		this.graf = graf;
		this.dg = dg;
		dg.mrizkaX = true;
		dg.mrizkaY = true;
	}

	public boolean vykresli(GC gc) {

		if(dg.stopa.get(0).getTypDat()!=3)
			return false;
		/// vykreslení mřížky
		if(dg.mrizkaY){
		gc.setLineStyle(SWT.LINE_DOT);
		for(int i = 0; i*dg.sM< dg.rec.width;i++){
			gc.drawLine(i*dg.sM + dg.rec.x, dg.rec.y, i*dg.sM +
dg.rec.x,
					dg.rec.y+dg.rec.height);

	}
		}
		if(dg.mrizkaX){
		for(int i = 0; i*dg.sM < dg.rec.height;i++){
			gc.drawLine(dg.rec.width+dg.rec.x,dg.rec.height+
dg.rec.y-(i*dg.sM),
					dg.rec.x,dg.rec.height+dg.rec.y-
(i*dg.sM));

	}
		gc.setLineStyle(SWT.LINE_SOLID);
		gc.setLineWidth(5);
		gc.drawLine(dg.rec.x,(int)(dg.rec.y+dg.rec.height-dg.nulaY),
dg.rec.width+dg.rec.x,
				(int)(dg.rec.y+dg.rec.height-dg.nulaY));
		gc.drawLine((int)(dg.rec.x+dg.nulaX), dg.rec.y+dg.rec.height,
				(int)(dg.rec.x +dg.nulaX), dg.rec.y);
		gc.setLineWidth(1);
		}
		for(Stopa s:dg.stopa){

	gc.setForeground(graf.getDisplay().getSystemColor(s.getBarva()));
		for(int i = 0 ; i< s.Xd.length-2 && i< s.Yd.length-2;i++){

				gc.setLineWidth(5);
			gc.drawLine((int)(dg.rec.x+dg.nulaX+(s.Xd[i]/dg.qx)),
					(int)(dg.rec.y+dg.rec.height-dg.nulaY-
(s.Yd[i]/dg.qy)),
					(int)(dg.rec.x+dg.nulaX+(s.Xd[i+1]/dg.qx)),
					(int)(dg.rec.y+dg.rec.height-dg.nulaY-
(s.Yd[i+1]/dg.qy)));
```

```
                }
                System.out.println("fff");
            }
        return true;
        }
}
```

Příloha J třída Stopa

```
package cz.martinvoracek.bp;

public class Stopa {
      private String nazev;
      private final int typDat;
      private int barva;
      private int typCary;
      public String X;
      public String Y;
      public double[] Xd;
      public double[] Yd; //Pole na hodnoty stopy

      public Stopa(String s,double dx[] ,String sy ){
            this.nazev = s;
            this.Xd = dx;
            this.Y = sy;
            this.typDat = 1;
      }
      public Stopa(String s,double dx[]){
            this.nazev = s;
            this.Xd = dx;
            this.typDat = 2;
      }
      public Stopa(String s,double dx[],double dy[]){
            this.nazev = s;
            this.Xd = dx;
            this.Yd = dy;
            this.typDat = 3;
      }

   public int getTypDat(){return this.typDat;}
   public String getNazev() {return nazev;}
   public void setNazev(String nazev) {this.nazev = nazev;}
   public int getBarva() {return barva;}
   public void setBarva(int barva) {this.barva = barva;}
   public int getTypCary() {return typCary;}
   public void setTypCary(int typCary) {this.typCary = typCary;}
}
```

Příloha K třída Title

```
package cz.martinvoracek.bp;

import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.graphics.Font;
import org.eclipse.swt.graphics.GC;
import org.eclipse.swt.graphics.Transform;

public class Title {

      private String nazev;
```

```
	private Color barva;
	private Font font;
	private Font fontOs;
	private DataGrafu dg;
	private Graf graf;

	Title(Graf graf,DataGrafu dg){
		this.graf = graf;
		this.dg = dg;
		this.font = new Font(graf.getDisplay(),dg.title.getPismo(),
				dg.title.getVelikos(),dg.title.getTyp());
		this.fontOs = new Font(graf.getDisplay(),
dg.title.getPismoXY(),
				dg.title.getVelikosXY(), dg.title.getTypXY());
		this.barva =
this.graf.getDisplay().getSystemColor(dg.title.getBarva());
	}
	public void vykresli(GC gc){
		this.nazev = dg.title.getText();
		gc.setFont(font);
		gc.setForeground(barva);
		gc.drawText(nazev,dg.rec.width/2 - gc.textExtent(nazev).x/2,
10);
		dg.rec.y = dg.rec.y + (gc.textExtent(nazev).y+20);
		dg.rec.height = dg.rec.height - dg.rec.y-10;
	}
	public void vykresliX(GC gc){
		this.nazev = dg.title.getTextX();
		gc.setForeground(barva);
		gc.setFont(fontOs);
		dg.rec.height = dg.rec.height -
gc.stringExtent(this.nazev).y;
		gc.drawText(this.nazev,dg.rec.x+dg.rec.width/2 -
gc.stringExtent(this.nazev).x/2,
				dg.rec.y + dg.rec.height);
	}
	public void vykresliY(GC gc){
		this.nazev = dg.title.getTextY();
		gc.setForeground(barva);
		gc.setFont(fontOs);
		dg.rec.x = dg.rec.x + gc.stringExtent(this.nazev).y;
		dg.rec.width = dg.rec.width - gc.stringExtent(this.nazev).y;
		Transform tr = new Transform(graf.getDisplay());
		tr.rotate(-90);
		gc.setTransform(tr);
		gc.drawText(this.nazev,-dg.rec.y - dg.rec.height/2 -
gc.stringExtent(nazev).x/2,
				dg.rec.x - gc.stringExtent(nazev).y);
		tr.rotate(90);
		gc.setTransform(tr);
	}
}
```

Příloha L třída TypGrafu

```
package cz.martinvoracek.bp;

import org.eclipse.swt.graphics.GC;

public interface TypGrafu {
```

```
      public boolean vykresli(GC gc);
}
```

Příloha N třída TypGrafu

```
package cz.martinvoracek.bp;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Font;
import org.eclipse.swt.graphics.GC;
import org.eclipse.swt.widgets.Canvas;

public class Legenda {

      private int x,y,sirka,vyska; //legendy
      private DataGrafu dg;
      private Canvas graf;
      private Font font;
      private int dy;

      Legenda(Graf graf,DataGrafu dg){
            font = new
Font(graf.getDisplay(),dg.legPismo,dg.legVelPisma,0);
            this.graf = graf;
            this.dg = dg;
      }
      public void vykresli(GC gc){
            this.vyska = 0;
            gc.setFont(this.font);
            for(Stopa s:dg.stopa){

                  if(gc.textExtent(s.getNazev()).x > this.sirka){
                        this.dy = gc.textExtent(s.getNazev()).y+5;
                        this.sirka = gc.textExtent(s.getNazev()).x +
this.dy+15;
                  }
                      this.vyska = this.vyska +
gc.textExtent(s.getNazev()).y+10;
                     gc.drawText(s.getNazev(), dg.rec.width -
sirka+dy+5,dg.rec.height/2  + vyska);


gc.setBackground(graf.getDisplay().getSystemColor(s.getBarva()));
                     gc.fillRectangle(dg.rec.width - sirka+5,
dg.rec.height/2  + vyska,
                                 gc.textExtent(s.getNazev()).y,
gc.textExtent(s.getNazev()).y);

gc.setBackground(graf.getDisplay().getSystemColor(SWT.COLOR_WHITE));
            }
                dg.rec.width = graf.getClientArea().width - sirka;
                this.x  = dg.rec.width;
                this.y = dg.rec.height/2 + vyska/3;
      }
}
```