2014

Cvičení pro programování a databáze

PODOROBNÝ PŘEHLED CVIČENÍ PRO PROGRAMOVÁNÍ A DATABÁZE PRO 4.ROČNÍK

LUKÁŠ KALČOK

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
Jihomoravský kraj
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

1 Obsah

2 Cvičení pro Programování a Databáze

Vypracoval: Mgr. Lukáš Kalčok

Počet hodin: 64 hodin (32 dvouhodinových cvičení)

2.1 Předmluva

Cvičení jsou koncipovány tak, aby studenti mohli postupně pracovat a vytvořit funkční editor obrázků, který podporuje jak maticové operace na obrazu tak i klasické úpravy obrazu. Postupnou prací se studenti seznamují s různými koncepty, které se v programování aplikace využívají. Práce na grafickém editoru donutí studenty implementovat i různé složitější algoritmy, na kterých si rozšiřují logické myšlení. Používaní různých už implementovaných struktur a vytváření svých vlastních se stane pro studenty běžnější a vytváření vlastních grafických komponent jim přiblíží funkčnost GUI. Za pomoci předpřipravených **JUnit** testů je umožněno studentům zkontrolovat si práci samostatně a seznámit se s **JUnit** testy. Cvičení probíhají v IDE **NetBeans** a používají **Java 1.6** a výše a **JUnit 4.0**. V době práce na projektu mají studenti k dispozici kompletní dokumentaci k projektu a mají teda i v čase nepřítomnosti pedagoga možnost pracovat na projektu samostatně. Takto se studenti seznamují s prací s dokumentací a nutností samostatné práce.

2.2 Cíl

- Cílem cvičení je seznámit žáky z větším projektem a jeho strukturou.
- Seznámit se s problematikou počítačové grafiky a návazností struktur v rámci větší aplikace.
- Funkční aplikace pro úpravu obrazu.
- Rozvoj algoritmického myšlení.
- Rozvoj samostatné práce a práce s dokumentací.

2.3 Učební pomůcky a didaktická technika

2.3.1 Učební pomůcky

2.3.1.1 Software

NetBeans IDE - Netbeans IDE je vývojové prostředí pro programování v různých jazycích, například Java, PHP, C a C++. Více informací o vývojovém prostředí je možné najít na http://netbeans.org/.

JUnit 4.0 je jednoduchý framework pro psaní testů. Jeho instalace není nutná, protože se už nachází v instalačním balíčku NetBeans IDE. Více informací můžete najít na http://junit.org/.

JDK 1.6 (Java Development Kit) a výše slouží ke spouštění Java aplikací a ke kompilaci kódu. Instalace JDK je nutná a není obsažena v NetBeans IDE.

Zkompilovaná hotová aplikace je studentům dostupná ve formě jar balíku, který mohou spustit a zkontrolovat si funkčnost svojí aplikace se zdrojovou.

2.3.1.2 Dokumenty

JavaDoc dokumentace k projektu je psaná v anglickém jazyku. Je to z důvodu, aby se studenti seznamovali s výrazy, které se v programátorské praxi běžně používají. Formulace nejsou nijak složité a jsou přímočaré, proto studentům stačí pokročilá znalost anglického jazyka.

Zdrojové kódy aplikace, spolu s JUnit testy, jsou dostupné pouze pedagogovi. Zkompilování těchto zdrojových souborů je možné vytvořit běžící aplikaci.

2.3.2 Didaktická technika

Dataprojektor, specializovaná učebna s počítačem pro každého žaka. Učitelský počítač.

2.4 Průběh výuky

Osobně se přikláním k tomu, aby na těchto cvičeních byl pedagog v roli pomocníka a kontrolora, ne v pozici vyučujícího, co vede frontální výuku. Studenti mají velice rozdílnou jak povahu, tak schopnosti a dovednosti a mnohdy i jinou

sociální a společenskou situaci, která vplývá na jejich rychlost. Postup výuky bych volil v první čtvrtině roku spíše ukázkou a později bych přecházel do více samostatné práce. Na začátku hodiny je nutné zopakovat, co již už má být z minula naimplementováno a osvětlil nové učivo nejdříve na příkladu, k čemu slouží praktická ukázka aplikace, potom rozborem kódu aplikace. Zpočátku je možné nechat studenty odepsat složitější části kódu z dataprojektoru, no tuto praktiku ke konci ročníku absolutně vyloučit.

2.5 Časový plán výuky

Výuka je rozdělena do 25 dvouhodinových cvičení, což dává vcelku 50 hodin výuky. Studenti jsou ve čtvrtém ročníku, mají maturitní týdny a v průběhu roku, v době svátků, odpadávaní hodiny. Pokud by ale učiteli zůstal čas, nic mu nebrání v tom, aby se třídou doprogramoval další funkce, které budou zajímavé a přínosné pro studenty. Celkový časový plán doporučuju dodržet, protože je navržen tak, aby bylo možné aplikaci vyvinout postupně, vidět jak aplikace roste a pochopit souvislosti v ní. Dále je učivo rozloženo ve výuce tak, aby se střídaly těžší a lehčí pasáže.

2.6 Doplňující informace

Kódy funkcí či tříd, které jsou zde uvedené, jsou ve své plné podobě. Je pravděpodobné, že v části, kdy jsou uvedeny, nemohou být implementovány v plném rozsahu. Je to z důvodu, že aplikace je postupně vyvíjena a její funkčnost se postupně zvětšuje. Některé třídy v tomto období ještě nemusí existovat.

3 Jednotlivá cvičení

3.1 Úvod do GUI

Studenti již mají zkušenosti s GUI, ale toto GUI vytvářeli za pomoci programového kódu, nikoliv za pomoci NetBeans GUI designéru. Seznámení s GUI designérem je pro ně nutné a je nutné je seznámit s tím, jakým způsobem je možné GUI měnit a co všechno je možné nastavovat.

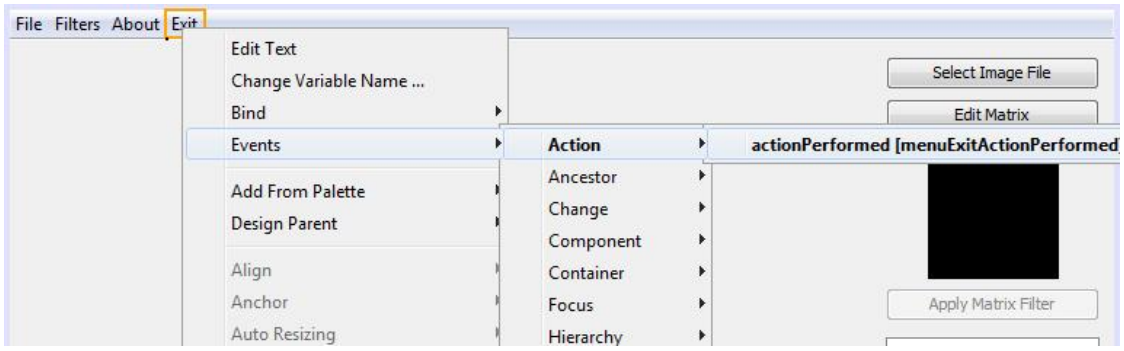
Modifikované třídy
`MainFrame`

Postup

1. Seznámení s vývojovým prostředím **NetBeans**.
    a. Zejména záložky ***File***, ***Window*** a ***Run***.
2. Vytvoření nového projektu.
    a. Přidávání nových `packages`.
    b. Ukázka nastavování projektu (cesty, parametry, hlavní třída …)
3. Vytvoření nové `Class`.
    a. Vytvoření `MainFrame` jako `Frame`.
    b. Ukázka přepínaní mezi obrazovkami ***design*** a ***source***.
    c. Přidání `JMenuBar, JMenu, JPanel, ButtonGroup, JRadioButton, JScrollPane,`
4. Ukázka bindování[1] funkcí a metodu pro tlačítko **Exit** v menu.

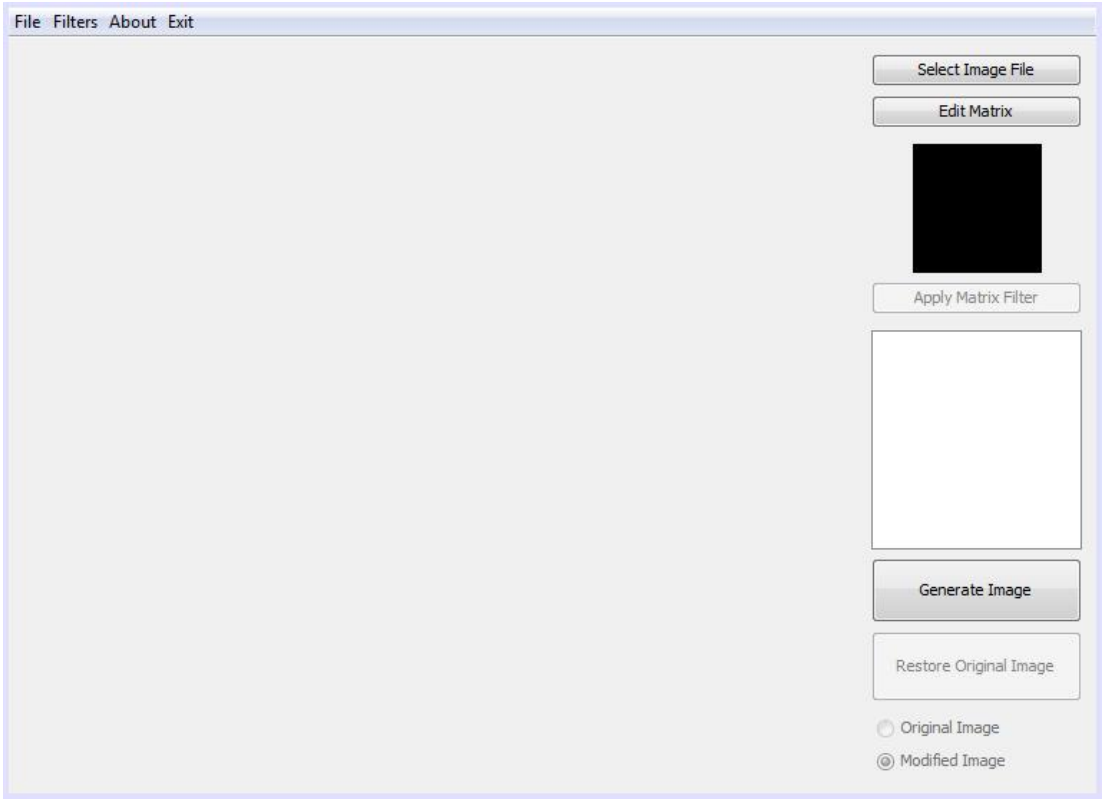
```
private void menuExitActionPerformed(java.awt.event.ActionEvent evt) {
      dispose();
}
```

Kód 1 Bindování metody dispose na tlačítko Exit



Obrázek 1 Ukázka bindování metody na událost v GUI

[1] Výraz používán pro spojení funkcí a GUI komponent.



Obrázek 2 Ukázka GUI aplikace

3.2 Načtení a ukládání obrázků

Studenti jsou seznámeni s `BufferedImage` a jejími možnostmi. Jsou taktéž seznámeni s používáním systémových oken, jako je například `JFileChooser` pro výběr obrázků a nastavovaní filtrů pro tyto obrazovky. Jako doplňková funkcionalita se naskýtá ověřování jména souboru pro ukládání. Ověřování koncovky souboru a případného doplnění koncovky. Na konci tohoto cvičení by měli být studenti schopni načíst soubor a uložit ho jako jiný soubor.

Modifikované třídy

`MainFrame`

Používané třídy

`JFileChooser` je GUI komponenta, která umožňuje prohledávat složky a vybírat soubory. Tato třída umožňuje nastavovat filtr pro zobrazování v době prohledávání disku. Třídu **`FileNameExtensionFilter`** je možno přímo za pomoci metody `setFileFilter` přiřadit instanci `JFileChooser`.

`BufferedImage` je základní struktura pro práci s obrazovou informací v Javě. Je možno v ní přímo nastavovat barvu pixlů a je možno za pomoci třídy `ImageIO` přímo načíst nebo uložit obrázek.

Důležité funkce

`loadImage()` – Metoda otevře `JFileChooser` a pokud si uživatel vybere obrázek, tak se pokusí v bloku `try` načíst obrázek. Pro žaky je ostatek kódu v této fázi zbytečný.

```
private void loadImage(){
    int returnVal = fileChooserLoad.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION){
        File file = fileChooserLoad.getSelectedFile();
        img = null;
        try {
            img = ImageIO.read(file);
            originalImage = img;
            printIntoLog("Loaded image:" + file.getName());
            redrawPanel();
            for (int a = 0; a < menuFilters.getItemCount(); a++){
                menuFilters.getItem(a).setEnabled(true);
            };
            buttonRunFilter.setEnabled(true);
            menuItemSaveImage.setEnabled(true);
            buttonRestoreOriginal.setEnabled(true);
            radioButtonOriginal.setSelected(true);
            radioButtonModified.setEnabled(true);
            radioButtonOriginal.setEnabled(true);
        } catch (IOException ex) {
            excDialog = new ExceptionDialog(this, true){};
            excDialog.setText("Error while reading file.");
            excDialog.setVisible(true);
        }
    }
}
```

Kód 2 Metoda loadImage

`saveImage()` – Metoda otevře `JFileChooser` a pokud si uživatel vybere obrázek, tak nejdříve zkontroluje, zdali má koncovku „jpg“ nebo „jpeg“ a pokud ne, tak ji dodá na konec názvu soboru. Do tohoto souboru se posléze pokusí uložit obrázek ve formátu JPEG. Kontrola názvu souboru probíhá, protože uživatel si mohl název napsat sám. Třída `FileNameExtensionFilter` pouze dodává filtr pro zobrazování a nedoplňuje automaticky koncovku souboru.

```
private void saveImage(){
   int returnVal = fileChooserSave.showSaveDialog(this);
   if (returnVal == JFileChooser.APPROVE_OPTION){
        File file = fileChooserSave.getSelectedFile();
        int i = file.getName().lastIndexOf('.');
        if (i > 0){
            String ext = file.getName().substring(i);
            ext = ext.toLowerCase();
            if (ext == null || ext != ".jpg" || ext != ".jpeg"){
                file = new File(file.getAbsoluteFile()+".jpg");
            }
        } else {
            file = new File(file.getAbsoluteFile()+".jpg");
        }
        try {
            ImageIO.write(img, "jpeg", file);
        } catch (IOException e){
             excDialog = new ExceptionDialog(this, true){};
             excDialog.setText("Error while writing file");
             excDialog.setVisible(true);
        }
    }
}
```

Kód 3 Zdrojový kód funkce saveImage

3.3 Práce s JavaDoc v NetBeans

Studenti mají dokumentaci k projektu, a proto je nutné seznámit je s tím, jak je možno dokumentaci generovat. Úloha učitele je, aby žáky seznámil s celou dokumentací daného projektu a ukázal jim, jak s ní manipulovat a jak ji využívat. Druhá část spočívá v tom, že jim ukáže, jak napsat komentáře v **Java** tak, aby po běhu **JavaDoc** byla vygenerována korektní dokumentace.

Postup

1. Seznámení s dokumentací, která byla studentům vložena. Postupné osvětlení všech tříd, které se v dokumentaci nachází. Ukázání, jakým způsobem je provázána dokumentace projektu s dokumentací k **Java**. Základy orientace v dokumentaci.
2. Vytvoření vlastní dokumentace.
    a. Přidání komentářů do funkcí.
    b. Názorná ukázka pro `@author`, `@link`, `@literal`, `@param`, `@return`, `@throws`, `@version`

```
/**
 * Main class of application. Runs all.
 *
 * @version 1.0
 * @author Lukas Kalcok
 */
```

Kód 4 Ukázka zdrojového kódu dokumentace

Více informací je možné najít na stránkách:
http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html#referenceguide

3.4 Vlastní panel pro zobrazování náhledu obrázku

Aplikace již už dokáže načíst a uložit obrázek, proto je vhodné zobrazovat ho. Třída `CustomJPanel` umožňuje obrázek zobrazit. Třída vznikla proto, aby bylo možné obrázek zobrazovat v reálném rozlišení nebo zmenšen na jednu polovinu, třetinu, čtvrtinu atd. Třída `CustomPanel` také zobrazí ve své instanci `JLabel`, který při zmenšení zobrazí varování pro uživatele, že se nejedná o plné rozlišení, ale pouze náhled. Na konci cvičení by měli mít všichni studenti po načtení obrázku zobrazen obrázek na předním panelu v instanci `CustomJPanel`.

Modifikované třídy

`MainFrame, CustomJPanel`

3.4.1 CustomJPanel

Důležité metody

CustomJPanel() – Konstruktor třídy v sobě vytváří nový `JLabel`, který napozicuje a přidá do sebe.

```
public CustomJPanel(){
   warningLabel = new JLabel(" ");
   warningLabel.setSize(150, 20);
   warningLabel.setLocation(10, 10);
   this.add(warningLabel);
}
```

Kód 5 Zdrojový kód konstruktoru třídy CustomJPanel

`generateSmallImage()` – Tato metoda zjišťuje, zdali je nutno vytvořit malý obrázek znovu nebo nikoliv. Toto je ověřováno porovnáním změny koeficientu při změně velikosti komponenty. Malý obrázek je vytvořen, pokud je proměnná `generateNewImage` nastavena na hodnotu `true`.

```
private void generateSmallImage(){
    int height = super.getHeight();
    int width = super.getWidth();
    int coefF = 1;
    if (image != null){
        while (height < (image.getHeight())/coefF){
            coefF++;
        }
        while (width < (image.getWidth())/coefF){
            coefF++;
        }
        if (coef != coefF || generateNewImage) {
            coef = coefF;
            resizeImage(coef, image);
            generateNewImage = false;
        }
    }
}
```

Kód 6 Zdrojový kód funkce generateSmallImage třídy CustomJPanel

`resizeImage(int coefficient, Buffered img)` – Metoda vytváří náhled obrázku ve zmenšení dané koeficientem. Tato implementace vezme pouze každý x-tý pixel a v současnosti nebere v úvahu žádný aliasing ani jinou metodu.

```
private void resizeImage(int coefficient, BufferedImage img){
   if (coefficient < 1) coefficient = 1;
   int height = (int)Math.floor(img.getHeight()/coefficient);
   int width = (int)Math.floor(img.getWidth()/coefficient);
```

```
    smallImage = new BufferedImage(width, height, img.getType());
    for (int x = 0; x < width; x ++){
       for (int y = 0; y< height; y++){
          smallImage.setRGB(x, y, img.getRGB(x*coefficient, y*coefficient));
       }
    }
}
```

Kód 7 Zdrojový kód metody resizeImage třídy CustomJPanel

`paintComponent(Graphics g)` – metoda za pomoci `@Override` přepisuje funkci, která je definována pro `JPanel`. Systém zavolá funkci vždy, když dojde k překreslení komponenty. Pokud se změní velikost komponenty, tak se zavolá metoda `generateSmallImage`, která se rozhoduje, jestli se bude vytvářet nový obrázek. Potom se na plochu komponenty nakreslí zmenšený obrázek. Dále, jestli je pomocí proměnné `coef` detekováno, že obrázek je zmenšen, vykreslí se varovný text.

```
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (lastHeight != super.getHeight() || lastWidth != super.getWidth()){
        generateSmallImage();
    }
    g.drawImage(smallImage, 0, 0, this);
    lastHeight = super.getHeight();
    lastWidth = super.getHeight();
    if (coef > 1){
        warningLabel.setText("WARNING: scale:" + coef);
        warningLabel.setForeground(Color.red);
    } else {
        warningLabel.setText(" ");
     }
}
```

Kód 8 Zdrojový kód funkce paintComponent třídy CustomJPanel

Další metody v této třídě jsou jasné z dokumentace.

3.4.2 Úprava MainFrame

V tomto stádiu je nutné jenom přidat do metody `loadImage` spuštění metody `redrawPanel`. Metoda `redrawPanel` se stará o nastavení obrázku pro panel a o překreslení daného panelu. Je zde nutné myslet na přetypování instance `JCusomPanel`.

Pro to, aby bylo možné mít `panel` jako instanci třídy `JCustomPanel`, je nutné změnit konstruktor. Toto NetBeans umožnuje pouze za pomoci „Customize Code …“, kterou je možno najít v menu po pravém stlačení myši po označení komponenty.

Obrázek 3 Náhled obrazovky Code Customizer v NetBeans

3.5 Generování obrázku

Protože opětovné načtení obrázků po každém spuštění programu je časově náročné, je vhodné vložit na přední obrazovku tlačítko generující obrázek. Takto se žáci seznámí podrobněji s `BufferedImage` a pochopí práci s obrazem.

Modifikované třídy:

`MainFrame`

Důležité metody

`generateImage()` – Když je metoda zavolána, tak zavolá funkci `makeColoredImage` a její výsledek nastaví jako originální obrázek.

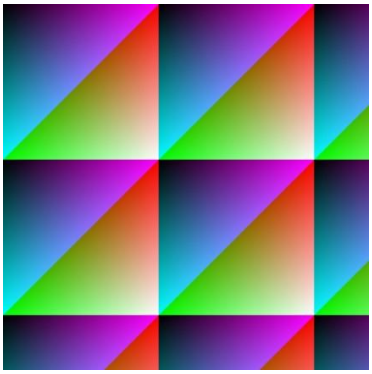
```
private void generateImage() {
    img = makeColoredImage();
    originalImage = img;
    printIntoLog("Image generated");
    redrawPanel();
    for (int a = 0; a < menuFilters.getItemCount(); a++){
        menuFilters.getItem(a).setEnabled(true);
    };
    buttonRunFilter.setEnabled(true);
    menuItemSaveImage.setEnabled(true);
    buttonRestoreOriginal.setEnabled(true);
    radioButtonOriginal.setSelected(true);
    radioButtonModified.setEnabled(true);
    radioButtonOriginal.setEnabled(true);
}
```

Kód 9 Zdrojový kód metody generateImage třídy MainFrame

`makeColoredImage()` – tato metoda za pomoci metody `%` (modulo) generuje obrázky o velkosti 600*600. Tato metoda vždy vygeneruje stejný obrázek. Neukazujte tuto funkci žákům a řekněte jim, ať se sami snaží za pomoci metoda modulo vytvořit obrázek. Je samozřejmé, že každý žák bude mít jiný obrázek. Dávejte na to pozor.

```
public BufferedImage makeColoredImage(){
    BufferedImage bImage = new BufferedImage(600, 600, BufferedImage.TYPE_3BYTE_BGR);
    for (int x = 0; x < bImage.getWidth(); x++){
       for (int y = 0; y < bImage.getHeight(); y++){
           bImage.setRGB(x, y, (new Color(x%255,y%255,(x+y)%255).getRGB()));
       }
    }
    return bImage;
}
```

Kód 10 Zdrojový kód metody makeColoredImage třídy MainFrame



Obrázek 4 Ukázka obrázku generovaného funkcí makeColoredImage

3.6 Vyjímky v Java a vytvoření filtru Identity

V tomto cvičení se studenti seznámí s pojmem výjimka v prostředí Java a jejich využití. Naučí se vytvořit si vlastní vyjímky, odchytávat je a reagovat na ně. Samotný `IdentityFilter` implementuje `ImageFilter`. Tento `ImageFilter` si studenti vytvoří za pomoci dokumentace, kterou k projektu mají. `ImageFilter` používá výjimky a studenti jsou nuceni tyto výjimky odchytávat za běhu a reagovat ně.

Modifikované třídy

`ImageFilter, IdentityFilter, MainFrame, FilterException`

Postup

1. `ImageFilter` – Pokud žáci vytvoří třídu `ImageFilter` podle dokumentace, tak následné vytváření `IdentityFilter` bude snadné.
2. `FilterException` – Pro běh `IdentityFilter` je nutné mít vlastní vyjímku `FilterException`. Tato vyjímka je využívána ihned ve `IdetityFilter` a je vyvolána pokud se snažíme pustit metodu `applyFilter` bez toho, abychom filtru nastavili obrázek, na kterém má filtr běžet.
3. `IdentityFilter` – Tato třída za pomoci deklarace `implements` implementuje všechny metody, které obsahuje interface `ImageFilter`.
4. `MainFrame` – Upravíme tuto třídu tak, aby bylo možné z menu pod záložkou **Filters** spustit daný `Identityfilter`.

3.6.1 ImageFilter

Metody v `ImageFilter` jsou z popisu dokumentace jednoznačné. Metoda `applyFilter` může ve všeobecnosti generovat vyjímku. Tato je generována samotnou třídou a jedná se o stavy, kdy například neexistuje obrázek, na který by se mněl filtr aplikovat nebo pokud nastane za běhu filtru jiná vyjímka. Tato vyjímka, například `DivisionByZero` je odchycena a nahrazena vyjímkou `FilterException`.

3.6.2 IdentityFilter

Metoda `applyFilter` vyhazuje výjimku, pokud se pokusí uživatel spustit ji a ještě není nastaven obrázek, na kterém by se měla spustit.

```
public void applyFilter() throws FilterException{
    if (originalImage == null){
        throw new FilterException();
    }
}
```

Kód 11 Zdrojový kód metody applyFilter třídy IdentityFilter

3.6.3 FilterException

Tato třída není výjimečná a je vytvořena pouze, abychom byli schopni reagovat na specifické vyjímky v našich filtrech.

3.6.4 Úprava MainFrame

Úprava samotného `MainFrame` pozůstává z přidání `JMenuItem` do `JMenu` s názvem `Filters`. Název `JMenuItem` je přidán manuálně a rutina na obsluhu po zmáčknutí dané položky v menu je vytvořena taktéž manuálně. V této době ještě nijak na vyhozenou výjimku v třídě `MainFrame` nereagujeme.

3.7 NegativeFilter

`NegativeFilter` je taktéž jako `IdentifyFilter` odvozen od `ImageFilter`, ale tento filtr vykonává negaci obrazu a to po složkách. Samostatně zpracuje R, G i B složku. Studenti naimplementují `ExceptionDialog`, který umožní při zachycení výjimky sdělit uživateli informace o tom, kde nastala vyjímka a za jakých podmínek.

Modifikované třídy

`NegativeFilter, ExceptionDialog, MainFrame`

Postup

1. `ExceptionDialog` – Vytvoření tohoto `JDialog` za pomoci GUI designéru. `ExceptionDialog` má ještě i metodu `setText`, která nastaví `JLabel` na ploše `JDialog` požadovaný text. `ExceptionDialog` je ve třídě `MainFrame` zanese přímo do konstruktoru a potom je pouze opakovaně vytvořen.
2. `NegativeFilter` – Tak, jako už implementovaný `IdentityFilter`, tak i `NegativeFilter` implementuje `ImageFilter`. `Negativefilter` ve své metodě `applyFilter` vykonává negaci obrázku po složkách.
3. Úprava `MainFrame` tak, aby bylo možné spustit filtr za pomoci menu v položce **Filters**. Zde je nutné pamatovat na to, aby bylo v případe výjimky vyvoláno okno `ExceptionDialog` s patřičnou vyjímkou.

3.7.1 NegativeFilter

Metoda `applyFilter` používá třídu `Color`. Nejdříve zjistí složky R, G, B a po jedné je odečte od maxima, které se v obraze nachází. V našem případě je toto maximum 255. Dále vytvoří novou barvu a zjistí z ní kód barvy ve formátu `interger`. Tuto hodnotu nastaví do nového obrázku `filteredImage`, který byl vytvořen tak, aby odpovídal nastavení `originalImage`. Pokud metoda vyvolá jakoukoliv výjimku, tak je tato vyjímka zachycena a je vyvolána nová vyjímka `FilterException` s informacemi s originální vyjímky.

```
public void applyFilter() throws FilterException{
    try {
        filteredImage = new BufferedImage(originalImage.getWidth(),
                            originalImage.getHeight(), originalImage.getType());
        for (int x = 0; x < originalImage.getWidth(); x++){
            for (int y = 0; y < originalImage.getHeight(); y++){
                int rgbOrig = originalImage.getRGB(x, y);
                Color c = new Color(rgbOrig);
                int r = 255 - c.getRed();
                int g = 255 - c.getGreen();
                int b = 255 - c.getBlue();
                Color nc = new Color(r,g,b);
                filteredImage.setRGB(x,y,nc.getRGB());
            }
        }
    } catch (Exception e){
        throw new FilterException(e.getMessage());
    }
}
```

Kód 12 Zdrojový kód metody applyFilter třídy NegativeFilter

3.7.2 MainFrame

```
catch (FilterException ex) {
    ExceptionDialog ed = new ExceptionDialog(null, true);
    ed.setText(ex.getMessage());
    ed.setVisible(true);
}
```

Kód 13 Ukázka možného zachycení vyjímky a reakce na ní

Obrázek 5 Ukázka aplikace filtru negativ na obrázek

3.8 Kontrolní hodina a řešení problémů

V průběhu prvních sedmi hodin studenti jistě narazili na množství problémů, které se jim nepodařilo samostatně nebo s kolegy vyřešit. Pokud studentům něco neběží, může to negativně ovlivnit další vývoj jejich aplikace a schopnost udržovat tempo se skupinou. Na této hodině doporučuju obcházet studenty jednotlivě a kontrolovat stav jejich projektu. Tak též mít zde individuální přístup a problém s nimi opravdu vyřešit. Tato kontrola je taktéž nutná pro nás, abychom se ujistili, že výklad a cvičení probíhají správně a studenti vnímají látku korektně a je v jejich schopnostech programovat požadovanou aplikaci.

3.9 Dolaďování GUI

Studenti často používají při debugovaní[2] a celkově v běhu programu obrázek, který se musí vždy po aplikaci filtru na obraz načíst znovu. Toto lze jednoduše obejít tím, že na přední obrazovku vložíme dvojici RadioButton-u, které nám umožní přepínat mezi originálním a modifikovaným obrázkem. Doladíme zpřístupňování či znepřístupňování tlačítek podle toho, jestli na ně uživatel může kliknout.

Modifikované třídy

`MainFrame`

Postup

Doporučuju studenty seznámit jenom s metodami komponent jako je `setEnabled`, `setSelected` a nechat je, ať se pokusí samostatně vyřešit problém, při kterém uživateli dovolí zmáčknout jenom tlačítka, které může v danou dobu stlačit.

[2] Výraz používaný v informatice a programování pro hledání chyb takzvaných bug-ů. Jak tento výraz vznikl je možné nalézt například zde: http://www.opensourceforu.com/2012/03/joy-of-programming-why-software-glitch-called-bug/

3.10 Ovladač filtrů

Třída `FilterController` nám umožní oddělit výkonnou část aplikace od části, která slouží pro ovládání aplikace. Doposud, pokud jsme chtěli spustit daný filtr, měli jsme ho přímo v Menu `MainFrame`. Nyní po implementaci třídy `FilterController` jsou položky v menu pod „Filters“ generovány na základě toho, co vrací instance třídy `FilterController`. `FiltrerControler` si sám ve svém konstruktoru vytvoří filtry a sám spouští dané filtry. Pokud třída `MainFrame` chce spustit filtr, tak zavolá v instanci `FilterController` metodu `applyfilter`, která spustí daný filtr.

Modifikované třídy

`MainFrame, FilterController, NoFilterException`

Postup

1. `NoFilterException` – Implementace této vyjímky je téměř identická jako v případě vyjímky `FilterException`.
2. `FilterController` – Tato třída má základní metody, které poskytují seznam filtrů a dovolují jejich běh.
3. Z menu odstraníme položky pro volání filtrů. Do třídy `MainFrame` je přidána funkce `loadFilters` a tato zabezpečuje, že je možné načíst dostupné filtry ve `FiltrController` a za pomoci přidání `ActionListener` je spouštět.

3.10.1 FilterController

`FilterController` má vlastní `Map`, který je implementován jako `HashMap`. V této struktuře se nachází přímo dvojice `String` a `ImageFilter`. Tato mapa je naplněna v konstruktoru třídy a to manuálně v kódu. Třída `FilterController` poskytuje za pomoci metody `getFilters` množinu dostupných filtrů. Metoda `runFilter` zabezpečuje běh daného filtru a reaguje na požadavek pro nedostupnost filtru vyjímkou `NoFilterException`. Vyjímku `FilterException` nijak nechytá a nechává třídu, která požadovala běh filtru na její zpracování. Metoda `runFilter` taktéž umožňuje rozeznat, jestli filtr není typu `JDialog` a v tomto případě zobrazí tento dialog a nezavolá metodu filtru `applyFilter`, ale zobrazí tento filtr za pomoci metody `setVisible` a nechá uživatele, aby mohl s daným oknem manipulovat sám.

```
private Map <String,ImageFilter> filters;
```

Kód 14 Struktura Map použitá pro uložení dostupných filtrů

```
public FilterController(Frame parent) {
    filters = new HashMap<>();
    NegativeFilter nf = new NegativeFilter();
    filters.put(nf.getFilterName(), nf);
    IdentityFilter iff = new IdentityFilter();
    filters.put(iff.getFilterName(),iff);
}
```

Kód 15 Konstruktor třídy FilterController

```
public BufferedImage runFilter(String name, BufferedImage image) throws
NoFilterException, FilterException{
   ImageFilter filter = filters.get(name);
   if (filter != null){
       filter.setImage(image);
       if (filter instanceof javax.swing.JDialog){
           ((JDialog)filter).setModal(true);
           ((JDialog)filter).setVisible(true);
       } else {
            filter.applyFilter();
       }
```

```
            return filter.getImage();
        } else {
            throw new NoFilterException(name);
        }
    }
```

Kód 16 Ukázka kódu metody runFilter třídy FilterController

3.10.2 Úprava MainFrame

Do konstruktoru třídy `MainFrame` vložíme vytvoření instance `FilterController` a na konec metody vložíme volání metody `loadFilters`. Tato metoda si vyžádá od instance `FilterController` množinu názvů filtrů a pro každý vytvoří samostatnou položku `JMenuItem` položku v menu „filters“. Pro to, aby bylo možné na dané tlačítka klikat a volat daný filtr pod daným jménem, je nutné přidat `ActionListener`. `ActionListener` je třída, které instance monitoruje, jestli se nevykonala akce na dané komponentě. Pokud ano, vyvolá se metoda `actionPerformed`. Tuto metodu vytváříme samostatně pro každý `jMenuItem` a je v ní pokus o zavolání metody `runFilter` třídy `FilterController`. Vše je ošetřeno blokem `try-catch` a pokud by nastala vyjímka typu `FilterException` nebo `NoFilterException`, tak je vyvoláno dialogové okno `ExceptionDialog`. Pokud nenastane vyjímka je za pomoci metodu `redrawPanel` překreslen obrázek.

```
private void loadFilters(){
  Set filters = filterController.getFilters();
  Iterator it = filters.iterator();
  while (it.hasNext()){
    JMenuItem jm = new JMenuItem((String)it.next());
    jm.setEnabled(false);
    jm.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        try {
          img = filterController.runFilter(((JMenuItem)e.getSource()).getText(), img);
          radioButtonModified.setSelected(true);
        } catch (FilterException | NoFilterException ex) {
          excDialog = new ExceptionDialog(null, true);
          excDialog.setText(ex.getMessage());
          excDialog.setVisible(true);
        }
        redrawPanel();
      }
    });
  menuFilters.add(jm);
  }
}
```

Kód 17 Zdrojový kód metody loadFilters třídy MainFrame

3.11 Filtr pro prahování obrázku

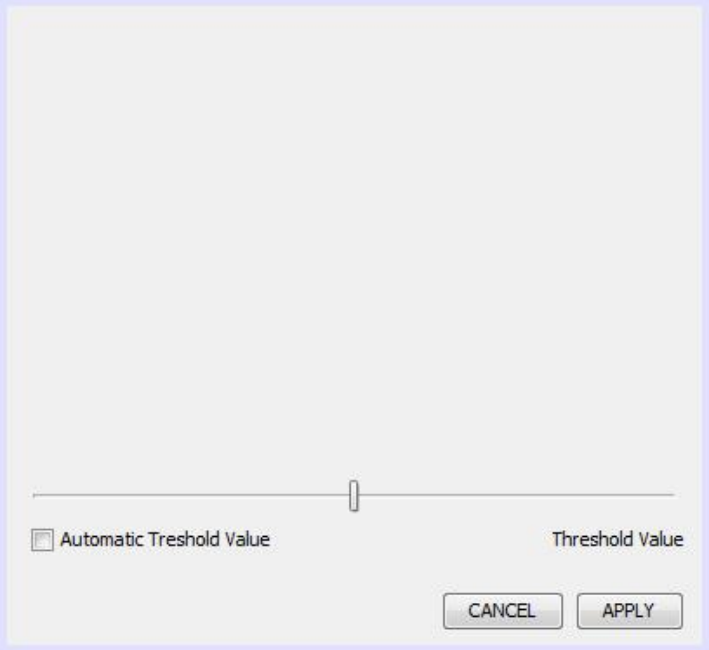
Metoda `ThresholdFilter` spočívá pouze v tom, že prahuje obrázek. V našem případě prahuje obrázek na velice zjednodušeně a to tak, že pokud jakákoliv hodnota z R, G nebo B přesahuje stanovený ***thresholdLimit***, tak je výsledná barva zkoumaného pixelu bíla, jinak je pixel černý. `ThresholdFilter` v našem podání umožňuje běh filtru automaticky. V tomto případě je hodnota ***threshold limit*** nastavena na 128 nebo je možno ho spustit jako dialogové okno. Dialogové okno obsahuje jak posuvník pro natavení threshold, tak zaškrtávací políčko pro automatické spočítání hodnoty ***threshold limit***. Taktéž obsahuje náhled výsledku filtru. Pro tento náhled je opět použita třída `CustomJPanel`. Zde se studenti seznámí s dědičností v Java a pojmy jako je `super` a `override`.

Modifikované třídy

`ThresholdFilter`

3.11.1 ThresholdFilter

`ThresholdFiler` je vytvořen jako `JDialog` a je mu přidána třída pro implementaci `ImageFilter`. Taktéž má táto třída metodu `applyFilter` s parametrem obrázek. V této třídě se nachází metody pro vstup uživatele jako je například posunutí `JSlider` nebo zaškrtnutí `JCheckBox` tlačítka pro automatické spočítání `currentTreshold`, což reprezentuje **threshold value**.



Obrázek 6 Ukázka návrhu GUI obrazovky pro ThresholdFilter

Metoda `getAutomaticThreshold` prochází obrázek a zjistí za pomoci průměru doporučený **threshold value**.

```
private int getAutomaticTreshold(){
    long outInt = 0;
    for (int x = 0; x < originalImage.getWidth(); x++){
        for (int y = 0; y < originalImage.getHeight(); y++){
            Color c = new Color(originalImage.getRGB(x, y));
            outInt = outInt + c.getRed() + c.getGreen() + c.getBlue();
        }
    }
    outInt = outInt / (originalImage.getHeight()*originalImage.getWidth()*3);
    return (int)outInt;
}
```

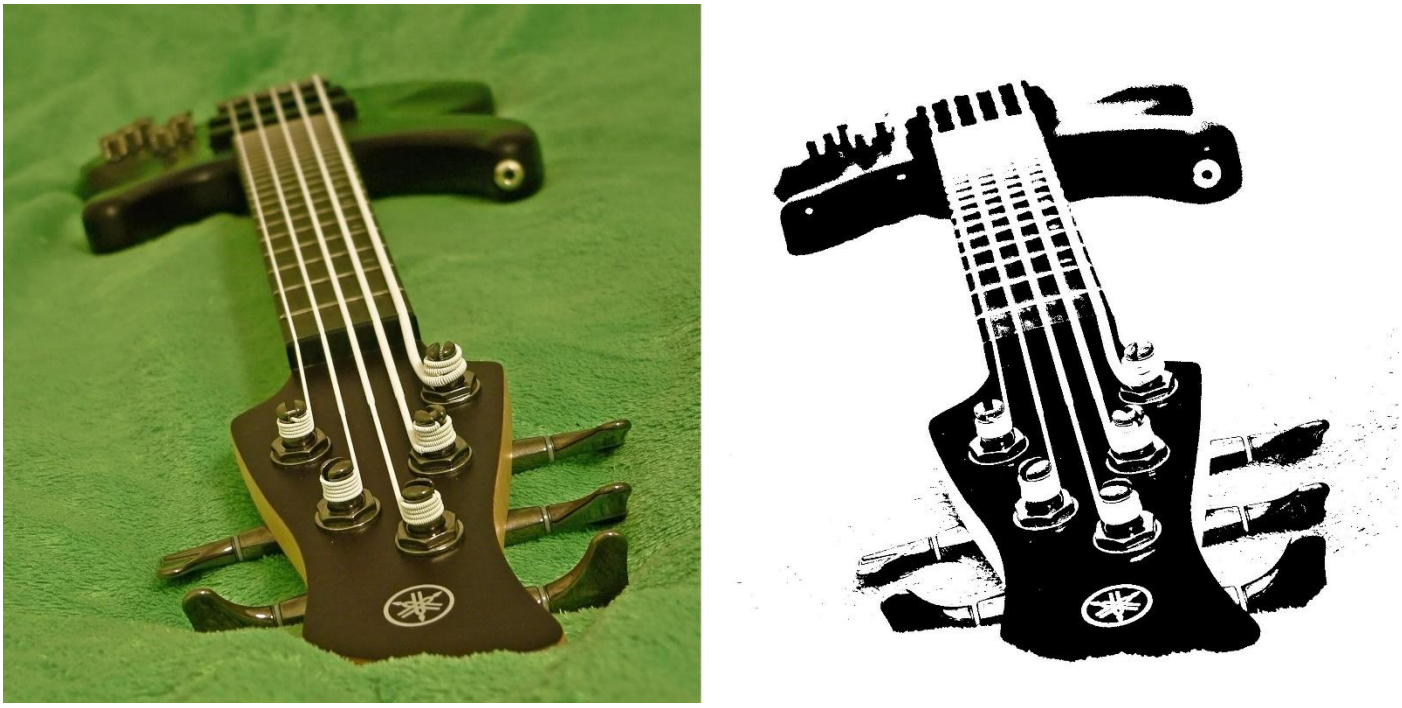
Kód 18 Zdfojový kód metody getAutomaticThreshold třídy ThresholdFilter

Metoda `applyFilter` s parametrem `BufferedImage` nahrazuje výkonnou část funkce `applyFilter` bez parametru. Metoda `applyFilter` bez parametru pouze volá tuto metodu. Takto je docíleno, že filtr je použitelný i bez toho, aby bylo nutné vyvolávat ho jako dialogové okno. V tomto případě se aplikuje prahování se základním nastavením hodnoty pro prahování. Pro překročení prahu stačí, aby kterákoliv složka RGB daného pixelu překročila limit nastavený uživatelem. Pokud je práh překročen, je daný pixel nastaven na bílou barvu, jinak je nastaven na černou.

```
private void applyFilter(BufferedImage img) throws FilterException{
   if (img == null) {
       throw new FilterException();
   }
   int black = Color.BLACK.getRGB();
   int white = Color.WHITE.getRGB();
   filteredImage = new BufferedImage(img.getWidth(),img.getHeight(),img.getType());
   for (int x = 0; x < img.getWidth(); x++){
      for (int y = 0; y < img.getHeight(); y++){
         int rgb = img.getRGB(x, y);
         Color c = new Color(rgb);
         if (c.getRed() > currentTreshold ||
             c.getGreen() > currentTreshold ||
             c.getBlue() > currentTreshold){
            filteredImage.setRGB(x, y, white);
         } else {
            filteredImage.setRGB(x,y,black);
         }
      }
   }
   ((CustomJPanel)panelImage).setImage(filteredImage);
   ((CustomJPanel)panelImage).repaint();
}
```

Kód 19 Zdrojový kód metody applyFilter třídy ThresholdFilter

Metody, které obsluhují posun `JSlider` nebo kliknutí na `JCheckBox` pro detekci a nastavení automatického **threshold value**, musí vygenerovaný obrázek. Toto je docíleno tím, že do instance `CustomJPanel` je vložen nový obrázek a je překreslen. O úpravu rozměrů se už stará třída `CustomJPanel`. Pokud se uživatel rozhodne, že nechce aplikovat daný filtr a klikne na tlačítko „Cancel“, tak se za pomoci proměnné `applyFilter` nastaví, že nemá být daný filtr aplikován na obraz. Metoda `dispose` se potom na základě této proměnné rozhodne, jestli vrátí původní obrázek nebo modifikovaný.



Obrázek 7 Ukázka prahování obrázku za pomoci ThresholdFilter

3.12 Matice pro budoucí filtr

Pro další filtr budeme potřebovat matici. Pro studenty teď stačí pouze informace o tom, že je to pole o velikosti X, Y a je plné `integer`. Matice však má více konstruktorů a různé další metody, které se mohou jevit jako problematické. Například metoda `getMatrixCopy` se pro studenty může jevit jako nedůležitá a je na ní možno osvětlit, jak fungují ukazovatele v Java.

Modifikované třídy

`ImageMatrix`

3.12.1 ImageMatrix

Třída `ImageMatrix` bude v budoucnu sloužit pro třídu `MatrixFilter`. Je to třída, i když by se dala nahradit strukturou. Třída nám umožňuje mít vlastní metody, které nemusíme potom v kódu tvořit vícenásobně ve více metodách v různých třídách. Třída umožňuje být vytvořena třemi způsoby a to pomocí základního konstruktoru nebo za pomoci konstruktorů, které buď definují velikost matice, nebo přímo definují naplnění matice. Za zmínku stojí metoda `getMatrixCopy`, která vytvoří kopii matice jako novou instanci a vrátí ji. Tato metoda je používána pro zkopírování matice do nového objektu a posléze zahození tohoto objektu. Práce s kopií nijak nemodifikuje původní matici, z které kopie vznikla.

3.13 Seznámení s JUnit.

JUnit je v současnosti nejpoužívanější nástroj na testování a ověřování správnosti kódu. Je vhodné studenty seznámit s testováním na nějakém vhodném příkladu. Osvědčilo se mi ukázat, jak se matematicky počítá Fibonacciho posloupnost jak rekurzivním, tak i nerekurzivním způsobem a pokusit se napsat test na tuto třídu. Ukázat studentům jak se dělá `@Test` a celý `testSuite`. Je třeba ukázat, jak vytvářet `assert` a taktéž jak testovat, jestli třída vyhodila očekávanou vyjímku na něčem, co mělo vyhodit vyjímku a použití `fail`.

3.13.1 Výpočet Fibonacciho čísla

Výpočet Fibonacciho čísla je definován matematicky jako:

$$fib(0) = 0$$

$$fib(1) = 1$$

$$fib(x) = fib(x-1) + fib(x-2), \qquad x \in N$$

Vzorec 1 Výpočet Fibonacciho čísla

Z tohoto výpočtu je možné odvodit i algoritmus pro spočítání daného čísla. V této ukázce jdou uvedeny obě metody výpočtu. Pro ukázku, jak vytvořit testování doporučuju vytvořit samostatný projekt v **NetBeans**. Tento projekt tvoří ze začátku jenom dvě třídy a to je `FibonacciCoding`, která obsahuje metodu `main` a slouží pro spouštění další třídy `FibonacciComputer`. Třída `FibonacciComputer` obsahuje konstruktor metody `computeFibonacciNumber` a `computeFibonacciNumberRecursive`. První metoda počítá Fibonacciho čísla pomocí pole, do kterého zapisuje čísla. Druhá metoda používá rekurzi a při požadavku na číslo se odvolává na sebe sama, ale s nižším číslem, které sečte a vrátí jako výsledek. Obě metody vyhazují vyjímku, pokud je zadáno číslo pro výpočet, které je menší než 0. Na tomto si žáci mohou vyzkoušet, že rekurzivní řešení problému je jednoduché na naprogramování, ale pro výpočet absolutně nevhodné. Pokud ve třídě `FibonaccCoding` nastavíme hodnotu `max` na 45 a více, studenti si mohou všimnout, že výpočet rekurzivním způsobem je mnohem pomalejší.

```
package fibonaccicoding;
public class FibonacciCoding {
    public static void main(String[] args) {
        FibonacciComputer fc = new FibonacciComputer();
        int max = 10;
        System.out.println("\n Compute Non Recursive");
        for (int x = 0; x < max; x++){
            try {
                System.out.print(fc.computeFibonacciNumber(x) + ",");
            } catch (Exception ex) {
                System.out.println(ex.getMessage());
            }
        }
        System.out.println("\n Compute Recursive");
        for (int x = 0; x < max; x++){
            try {
                System.out.print(fc.computeFibonacciNumberRecursive(x) + ",");
            } catch (Exception ex) {
                 System.out.println(ex.getMessage());
            }
        }
    }
}
```

Kód 20 Třída FibonacciCoding

```
package fibonaccicoding;
public class FibonacciComputer {
    public long computeFibonacciNumber(int index) throws Exception{
```

```
        if (index < 0){
            throw new Exception("Unable to compute fibonacci number below 0.");
        }
        long numbers[] = new long[index+2];
        numbers[0] = 0;
        numbers[1] = 1;
        for (int x = 2; x <= index; x++){
            numbers[x] = numbers[x-1] + numbers[x-2];
        }
        return numbers[index];
    }
    public long computeFibonacciNumberRecursive(int index) throws Exception{
        if (index < 0){
            throw new Exception("Unable to compute fibonacci number below 0.");
        }
        if (index == 0){
            return 0;
        }
        if (index == 1){
            return 1;
        }
        return    computeFibonacciNumberRecursive(index - 1) +
                  computeFibonacciNumberRecursive(index - 2);
    }
}
```

Kód 21 Třída FibonacciComputer

3.13.2 Testování výpočtu Fibonacciho čísla

Pro testování třídy `FibonacciComputer` je nutné vytvořit novou testovací `Suite`. Tento `Suite` nám umožňuje spouštět dané `class`, ve kterých jsou napsány testy. Samotné testy jsou v našem případě vytvořeny jako **JUnit 4.0** testy. Ve třídě `FibonacciCodingTestSuite` je za pomoci klauzule `@Suite.SuiteClasses({})` přidán seznam všech `class`, které se mají spouštět jako testy.

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses({FibonacciComputerTest.class})
public class FibonacciCodingTestSuite {}
```

Kód 22 Třída FibonacciCodingTestSuite

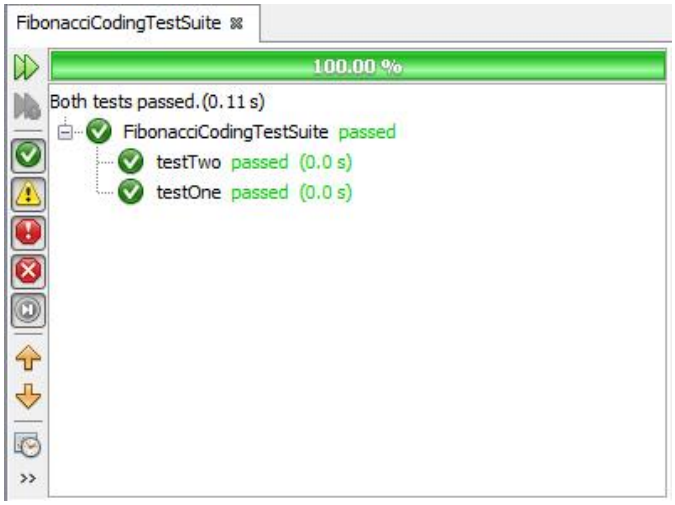
Třída `FibonacciComputerTest` obsahuje metody `setUpClass` a tearDownClass označené jako `@BeforeClass` a `@AfterClass`. Tyto metody se vykonají před spuštěním a po dokončení testu. Dále jsou zde metody `testOne` a `testTwo`, které jsou pomocí klauzule @Test definovány jako testy, které se mají vykonat. Metoda `testOne` s pomocí funkce `assertEquals` testuje, zda metody `computeFibonacciNumber` a `computeFibonacciNumberRecursive` počítají Fibonacciho čísla správně. Metoda `testTwo` testuje, jestli metody reagují na výpočet záporného Fibonacciho čísla vyhozením vyjímky. Pokud ne, tak pomocí funkce `fail` je test vyhodnocen jako neúspěšný.

```
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import fibonaccicoding.FibonacciComputer;
public class FibonacciComputerTest {
    public FibonacciComputerTest() {}

    @BeforeClass
    public static void setUpClass() {
```

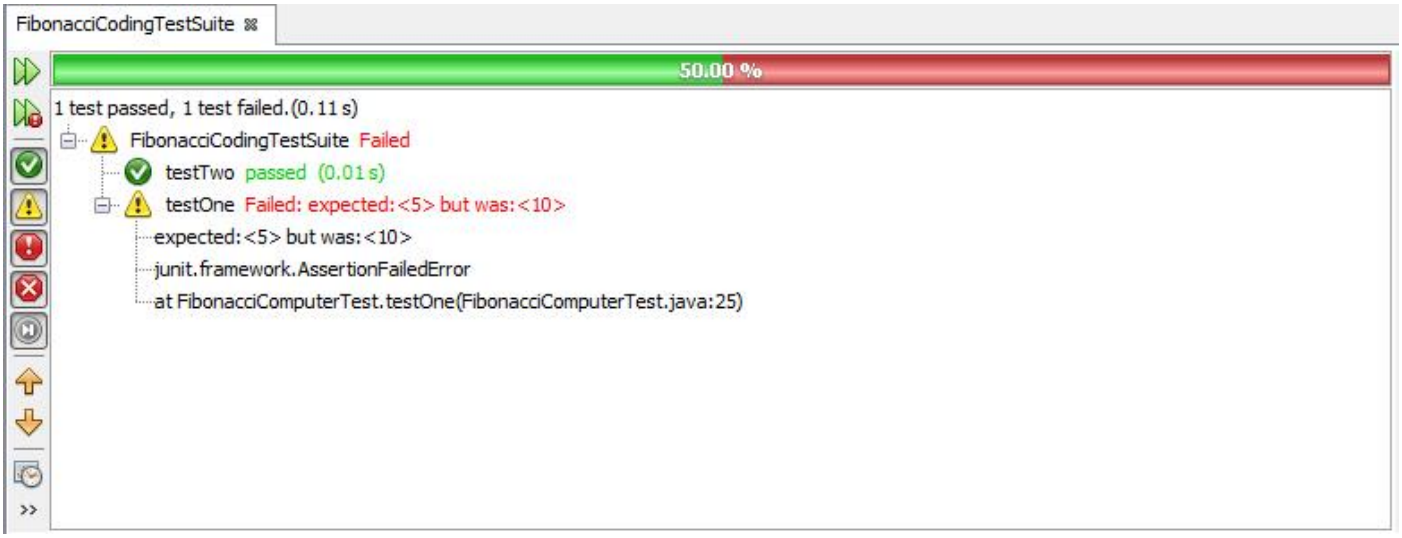
```
        System.out.println("Testing FibonacciComputer class.");
    }

    @AfterClass
    public static void tearDownClass() {
        System.out.println("End of testing FibonacciComputer class.");
    }

    @Test
    public void testOne() {
        FibonacciComputer fc = new FibonacciComputer();
        try {
            assertEquals(5, fc.computeFibonacciNumber(5));
            assertEquals(5, fc.computeFibonacciNumberRecursive(5));
        } catch (Exception ex) {
            fail();
        }
    }

    @Test
    public void testTwo(){
        FibonacciComputer fc = new FibonacciComputer();
        try {
            fc.computeFibonacciNumber(-10);
            fail();
        } catch (Exception ex) {}
        try {
            fc.computeFibonacciNumberRecursive(-10);
            fail();
        } catch (Exception ex) {}
    }
}
```

Kód 23 Třída FibonacciComputerTest

Po spuštění testů je v **IDE NetBeans** vidět, jak úspěšný byl běh dané `Suite`. Na základě této obrazovky je možné zjistit, kde nastala chyba a na čem daný test selhal.



Obrázek 8 Ukázka hladkého průběhu testovací suite JUnit



Obrázek 9 Ukázka nezdařeného běhu testovací suite JUnit

3.14 Zkouška dodaných testů na vlastních příkladech

Studentům je dodána testovací sada testů a je jim ukázáno, jak si je naimportovat do svého projektu. Studenti nyní zjistí, že je velice jednoduché detekovat, jestli mají daný kód správně napsaný nebo nikoliv. Tuto hodinu doporučuju vést velice individuálním přístupem a žákům ukazovat, kde jim testy hlásí chyby a proč. Poskytovat možnosti řešení, ale neopravovat kód, pokud to není nezbytně nutné.

Dodané třídy

`CustomJPanelTest, FilterControllerTest, GraphicsTestSuite, IdentityFilterTester, MatrixTester, NegativeFilterTester, NoFilterExceptionTester, ThresholdFilterTester, FilterExceptionTester`

3.15 Kontrolní den

Odevzdané úkoly pedagog zkontroluje a identifikuje nejčastější chyby, které se v pracích nacházely. Tyto chyby následně analyzuje před žáky a osvětlí, proč jsou dané postupy zvolené nevhodně nebo naopak ukáže řešení, které tyto chyby obchází.

3.16 Návrh obrazovky MatrixWindow

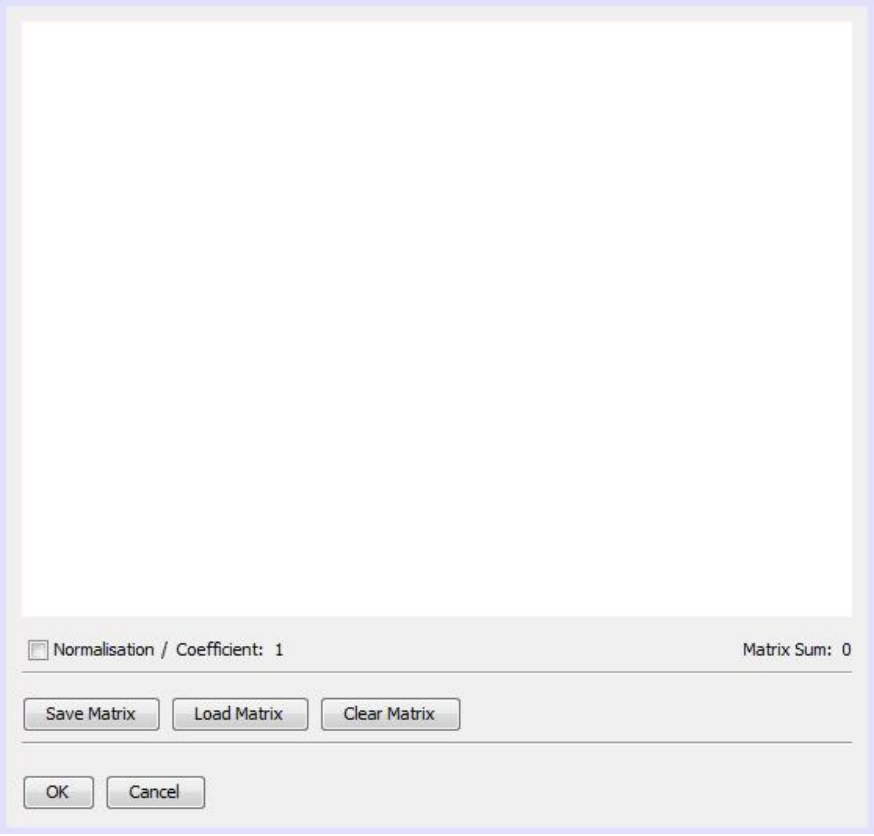
Studenti mají již vytvořenou třídu `ImageMatrix`. Tato třída obsahuje matici, která má určité rozměry a parametry `coefficient` a `normalized`. Tyto parametry je možno pomocí obrazovky `MatrixWindow` měnit. Obrazovka obsahuje kromě zaškrtávacího políčka pro zaznamenání normalizace taktéž tlačítka pro uložení matice, načtení a vyčištění matice matici samotnou. Tato je reprezentována za pomoci množství `JSpinner`, které jsou umístěny na ploše panelu. Generování těchto komponent je na základě vložené matice. Na konci tohoto cvičení by měl mít student GUI `MatrixWndow`, které se programově generuje podle vložené `ImageMatrix`. Tato obrazovka se upravuje podle toho, jak uživatel upravuje hodnoty v `JSpinner` nebo zaškrtává zaškrtávací políčko *Normalisation*.

Modifikované třídy

`MatrixWindow, MainFrame`

3.16.1 MatrixWindow

Třída `MatrixWindow` je typy `JDialog` a umožňuje úpravu `imageMatrix`. Pomocí automatického generování instancí třídy `JSpinner` vyplní pole `JPanel` na základě velikosti matice. Zaškrtávací tlačítko *„Normalisation“* umožňuje definovat, jestli je matice normalizovaná a do výpočtu pro filtr `MatrixFilter` se má počítat s koeficientem, který je ukazován jako textová informace. Textová informace *„Matrix Sum“* ukazuje celkovou velikost matice v součtu. Tlačítka *„Save Matrix“* a *„Load Matrix“* umožňují za pomoci třídy `XMLMatrixHandler` uložit a načíst matici ze souboru. Tlačítko *„Clear Matrix“* smaže matici a nastaví hodnotu koeficientu na 0, přičemž matice má v sobě uloženo, že výpočet s ní má probíhat bez normalizace.



Obrázek 10 Ukázka obrazovky MatrixWindow

Konstruktor třídy nejdříve zavolá pomocí `super` konstruktor `JDialog` a posléze zkopíruje do svých lokálních proměnných hodnoty vložené matice. Potom vytvoří pole `spinnerFields` a nechá proběhnout inicializaci GUI komponent. Dále vytvoří na panelu instance třídy `JSpinner`, které vloží do pole `spinnerFields`. Následně rozdistribuuje matici do daných `JSpinner`, zapíše hodnotu koeficientu a případně zaškrtne tlačítko *„Normalisation“*.

```
public MatrixWindow(java.awt.Frame parent, boolean modal, ImageMatrix m) {
   super(parent, modal);
```

```
    originalMatrix = m.getMatrixCopy();
    localMatrix = m.getMatrixCopy();
    spinnerFields = new JSpinner[localMatrix.length(0)][localMatrix.length(1)];
    initComponents();
    createFieldComponents();
    distributeMatrix();
    fileChooserLoad = new JFileChooser();
    fileChooserSave = new JFileChooser();
}
```

Kód 24 Konstruktor třídy MatrixWindow

Metoda `createFieldComponents` vytváří na základě `localMatrix` pole `JSpinner`, které vkládá do `JPanel`, který je zobrazován na ploše `MatrixWindow`. Taktéž přidává každé instanci `JSpinner ChangeListener`, který reaguje na změnu stavu této instance tak, že zavolá metodu `computeMatrixAndUpdateGUI`. Ke svému konci metoda validuje panel a překreslí ho.

```
private void createFieldComponents(){
    Border border = BorderFactory.createMatteBorder(1, 5, 1, 1, Color.gray);
    spinnerFields = new JSpinner[localMatrix.length(0)][localMatrix.length(1)];
    for (int x = 0; x < localMatrix.length(0); x++){
        for (int y = 0; y < localMatrix.length(1); y++){
            spinnerFields[x][y] = new JSpinner();
            spinnerFields[x][y].setSize(50, 30);
            spinnerFields[x][y].setBorder(border);
            spinnerFields[x][y].setLocation(60*x, y*40);
            spinnerFields[x][y].setVisible(true);
            spinnerFields[x][y].addChangeListener(new ChangeListener() {
                @Override
                public void stateChanged(ChangeEvent e) {
                    computeMatrixAndUpdateGUI();
                }
            });
            panel.add(spinnerFields[x][y]);
        }
    }
    panel.validate();
    panel.repaint();
}
```

Kód 25 Metoda createFieldComponent třídy MatrixWindow

Poslední poměrně zajímavou metodou je `computeMatrixAndUpdateGUI`. Tato metoda upraví matici na základě nastavení GUI podle uživatele a případně přepíše hodnoty koeficientu a celkové sumy matice.

```
private void computeMatrixAndUpdateGUI(){
  if (localMatrix.length(0) != spinnerFields.length ||
      localMatrix.length(1) != spinnerFields [0].length){
    clearFieldComponents();
    createFieldComponents();
  }
  for (int x = 0; x< localMatrix.length(0); x++){
    for (int y = 0; y < localMatrix.length(1); y++){
      localMatrix.set(x,y,(Integer)spinnerFields[x][y].getValue());
    }
  }
  labelOverall.setText(" " + ((Integer)getSum()).toString());
  localMatrix.setNormalization(checkboxNormalisation.isSelected());
  if (localMatrix.normalised()){
    localMatrix.setCoefficient(getCoefficient());
    labelNormalisation.setText(String.valueOf(localMatrix.getCoefficient()));
```

```
  } else {
    labelNormalisation.setText("Not Normalized");
  }
  localMatrix.setCoefficient(getCoefficient());
}
```

Kód 26 Metoda computeMatrixAndUpdateGUI třídy MatrixWindow

Další metody jsou určeny pro ukládání a načtení matice za pomoci `XMLMatrixHandler` nebo jsou určeny k zobrazení okna a vrácení matice. Zde je taktéž metoda `dispose` přetížena a chová se podle typu ukončení okna.

3.16.2 MainFrame

Změna ve třídě `MainFrame` je pouze v tom, že je nutné přidat vyvolání `MatrixWindow`.

```
private void buttonEditMatrixActionPerformed(java.awt.event.ActionEvent evt) {
    if (matrixWindow == null){
        matrixWindow = new MatrixWindow(this, true, matrix);
    }
    matrixWindow.setMatrix(matrix);
    matrix = matrixWindow.showDialog(true);
    drawMatrix();
    printIntoLog("New Matrix Set");
}
```

Kód 27 metoda pro obsluhu tlačítka Edit Matrix třídy MainFrame

3.17 Vytvoření kontrolní obrazovky pro zobrazení matice

Pro kontrolu a zobrazení aktuální matice je nutné vytvořit třídu `MatrixJPanel`. Její úlohou je zobrazovat matici, která je do ní vložena za pomoci barevného rozlišení. Tento panel je umístěn na ploše `MainFrame` a je obnoven pokaždé, když nastane změna matice.

Modifikované třídy

MatrixJPanel, MainFrame

3.17.1 MatrixJPanel

Třída je potomek `JPanel` a obsahuje metody `setMatrix`, která nastaví matici pro zobrazování a `paintComponent`, která zabezpečuje překreslování dané matice. Metoda `paintComponent` spočítá, kolik pixelů reprezentuje jednu položku v matici a dynamicky se rozhodne podle maxima a minima v matici jakou barvou jsou reprezentovány jednotlivé hodnoty v matici. V tomto konkrétním příkladu se pohybuje barevná škála v odstínech červené. Studenti si však mohou zvolit jakoukoliv barevnou škálu, pokud bude dostatečně jasné z vizualizace, jak je matice rozložená.

```
protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (matrix != null){
            int height = super.getHeight();
            int width = super.getWidth();
            int pixelPerBlockHeight = Math.round(height/matrix.length(0));
            int pixelPerBlockWidth = Math.round(width/matrix.length(1));
            int mMax = 0;
            int mMin = 0;
            for (int x = 0; x < matrix.length(0); x++){
                for (int y = 0; y < matrix.length(1); y++){
                    mMax = Math.max(mMax,matrix.get(x,y));
                    mMin = Math.min(mMin,matrix.get(x,y));
                }
            }
            int div = Math.max(mMax, Math.abs(mMin));
            int step = 0;
            if (div != 0){
                step = 127/(div+1);
            }
            int sizeX = WIDTH;
            int sizeY = HEIGHT;
            Color c;
            for (int x = 0; x < matrix.length(0); x++){
                for (int y = 0; y < matrix.length(1); y++){
                    if (matrix.get(x,y) == 0){
                        c = new Color(128,0,0);
                    } else {
                        c = new Color(128 + matrix.get(x,y)*step,0,0);
                    }
                    g.setColor(c);
                    g.fillRect( x*pixelPerBlockWidth, y*pixelPerBlockHeight,
                                pixelPerBlockWidth, pixelPerBlockHeight );
                }
            }
        } else {
           Color c = new Color(128,0,0);
           g.fillRect(0,0,super.getWidth(),super.getHeight());
        }
    }
}
```

Kód 28 Metoda paintComponent třídy MatrixJPanel

Na obrázku níže je možné vidět, jak se zobrazí níže uvedena matice v komponentě `MatrixJPanel`.

-1	**-2**	**-1**
0	**2**	**0**
1	**2**	**1**

Obrázek 11 Matice pro zobrazení v MatrixJPanel

Obrázek 12 Ukázka zobrazení matice v okně MatrixJPanel

3.17.2 MainFrame

Třída `MainFrame` zabezpečuje, aby po každé změně matice nastalo překreslení dané matice. Tato změna může nastat pouze v okně `MatrixWindow`. Pro toto stačí instanci `MatrixJPanel` vložit novou matici a zavolat ručně překreslení.

3.18 XML a DOM, práce ze souborem, zápis a čteni XML v Java

Studenti mají v tuto dobu už připravenou obrazovku pro úpravu matice. Uložení matice je poměrně triviální záležitost, pokud je programátor seznámen s třídami `DocumentBuilderFactory`, `DocumentBuilder` a `Document`. Metoda `loadMatrix` používá tyto třídy záměrně. Metoda `saveMatrix` záměrně tyto třídy nepoužívá, protože student by měl být schopen své XML uložit i manuálně do souboru za pomoci `BufferedWriter`. Výběr souboru je podobně jako v případě načtení obrázku docílen třídou `JFileChoooser`.

Modifikované třídy:

`XMLMatrixHandler, MatrixWindow`

3.18.1 XML a DOM

Extended Markup Language je značkovací jazyk určen zejména pro uložení dat společně s jejich strukturou. Byl vyvinut konsorciem W3C[3] a jeho současné rozšíření je celosvětové a používání je běžné. Tento jazyk se skládá z objektů nazývaných element, které mohou mít další elementy nebo atributy. Ukázku XML dokumentu je možné vidět níže. Elementy v níže zapsaném kódu jsou `recept`, `title`, `material`, `instructions` a `step`. Elementy v XML jsou vždy párové a počáteční `tag` je uveden jako `<…>` a koncový jako `</…>`. Atributy jsou uvedeny jako název `atributu="hodnota"`. Atributy jsou vždy navázaný na element, ve kterém se přímo nachází. V našem případě jde například o atribut `name`, kterého hodnota je `pancakes`. Tento atribut je navázán přímo na element `recept`.

```
<recept name="pancakes" rating="4" timeToComplete="2 hours">
  <title>Strawberry Pancakes</title>
  <material quantity="3" unit="cup">flower</material>
  <material quantity="2" unit="egg">eggs</material>
  <material quantity="0.050" unit="kg">sugar</material>
  <material quantity="0.5" unit="liter">milk</material>
  <material quantity="0.25" unit="kg">strawberry</material>
  <instructions>
    <step>Mix eggs, milk, sugar and flour in a bowl.</step>
    <step>Pour about 50g of mixed ingredients onto heated frying pan.</step>
    <step>Wait until the pancake is done.</step>
    <step>Put shredded strawberries onto pancake.</step>
    <step>Repeat until mixed ingredients are avalaible.</step>
  </instructions>
</recept>
```

Kód 29 Ukázka XML zápisu repceptu na jahodové palačinky

Tento dokument je možné si představit jako DOM, zkratka pro *Document Object Model*. Je to struktura, která má v sobě uloženou strukturu daného XML souboru. Umožňuje v něm vyhledávat a měnit ho.

```
recept | - title
       | - material
       | - material
       | - material
       | - material
       | - material
       | - instructions | - step
                        | - step
                        | - step
                        | - step
                        | - step
```

Kód 30 Ukázka struktury XML souboru s receptem pro výrobu palačinek

[3] W3C je zkratka pro *WorldWideWeb Consorcium*. Více informací je možné nalézt na stránkách http://www.w3.org/.

3.18.2 XMLMatrixHandler

Tato třída slouží k obsluze zápisu a načtení uložené matice. Její dvě hlavní metody jsou `saveMatrix` a `loadMatrix`. Metoda `saveMatrix` využívá zápisu do souboru skrz třídu `BufferedWriter` a zapisuje matici po řádcích. Metoda `loadMatrix` čte matici za pomoci třídy `DocumentBuilderFactory`, která společně s třídami `DocumentBuilder` a `Document` umožňuje vytvořit DOM nad daným XML souborem. Tento přístup je mnohem vhodnější než manuální parsování XML souboru, ve kterém bychom museli nad každým řádkem souboru řešit složitý algoritmus rozhodování. Validitu daného souboru za nás taktéž řeší použité třídy.

```
public void saveMatrix(File f) throws TransformerConfigurationException,
                                      TransformerException, IOException{
  if (matrix == null){
     throw new IIOException("Matrix is not set.");
  }
  if (!f.getName().endsWith(".xml")){
     f = new File(f.getPath() + ".xml");
  }
  f.createNewFile();
  try (BufferedWriter writer = new BufferedWriter(new FileWriter(f))) {
     writer.write("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\" ?>\n");
     writer.write("<matrix>\n");
     writer.write("  <dimensionX>" + matrix.length(0) + "</dimensionX>\n");
     writer.write("  <dimensionY>" + matrix.length(1) + "</dimensionY>\n");
     writer.write("  <normalised>" + matrix.normalised() + "</normalised>\n");
     writer.write("  <coefficient>" + matrix.getCoefficient() + "</coefficient>\n");
     for (int x = 0; x < matrix.length(0); x++){
       writer.write("  <column>\n");
       for (int y = 0; y < matrix.length(1); y++){
         writer.write("      <cell>" +
                      Integer.toString(matrix.get(x, y)) +
                      </cell>\n");
       }
       writer.write("  </column>\n");
     }
     writer.write("</matrix>\n");
  }
}
```

Kód 31 Metoda saveMatrix třídy XMLMatrixHandler

Metoda `loadMatrix` nejdříve načte pomocí tříd `NodeList` parametry matice jako je šířka, výška, koeficient či normalizace. Dále prochází všechny sloupce matice a načte jednotlivé hodnoty. Pokud zjistí, že je více hodnot pro sloupec, než výška matice, nebo je více sloupců, než šířka matice vyhodí vyjímku. Metoda nereaguje na menší množství dat.

```
public ImageMatrix loadMatrix(File f) throws ParserConfigurationException,
                                              SAXException, IOException{
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder db = dbf.newDocumentBuilder();
  Document doc = db.parse(f);
  doc.getDocumentElement().normalize();
  NodeList dimensionXNode = doc.getElementsByTagName("dimensionX");
  int dimX = new Integer(dimensionXNode.item(0).getTextContent());
  NodeList dimensionYNode = doc.getElementsByTagName("dimensionY");
  int dimY = new Integer(dimensionYNode.item(0).getTextContent());
  NodeList coefficientNode = doc.getElementsByTagName("coefficient");
  Float coefficient = new Float(coefficientNode.item(0).getTextContent());
  NodeList normalisedNode = doc.getElementsByTagName("normalised");
  String norm = normalisedNode.item(0).getTextContent();
  boolean nor = false;
  if (norm.equals("true")){
```

```
    nor = true;
  }
  matrix = new ImageMatrix(dimX, dimY, nor);
  matrix.setCoefficient(coefficient);
  matrix.setNormalization(nor);
  NodeList cols = doc.getElementsByTagName("column");
  if (cols.getLength() > dimX){
    throw new IOException("\"Bad Input format of XML file, Too much data. / X");
  }
  for (int x = 0; x < dimX; x++) {
    Node node = cols.item(x);
    Node cell = node.getFirstChild();
    int y = 0;
    while (cell != null){
      if (cell.getNodeName().equals("cell")){
        if (y >= dimY){
          throw new IOException("Bad Input format of XML file, Too much data.");
        }
        matrix.set(x, y, new Integer(cell.getTextContent()));
        y++;
      }
      cell = cell.getNextSibling();
    }
    if (y != dimY){
      throw new IOException("Bad Input format of XML file, Not enough data.");
    }
  }
return matrix;
```

Kód 32 Metoda loadMatrix třídy XMLMatrixHandler

```
<matrix>
  <dimensionX>3</dimensionX>
  <dimensionY>3</dimensionY>
  <normalised>true</normalised>
  <coefficient>0.5</coefficient>
  <column>
    <cell>-1</cell>
    <cell>0</cell>
    <cell>1</cell>
  </column>
  <column>
    <cell>-2</cell>
    <cell>2</cell>
    <cell>2</cell>
  </column>
  <column>
    <cell>-1</cell>
    <cell>0</cell>
    <cell>1</cell>
  </column>
</matrix>
```

Kód 33 Ukázka zápisu matice o velikosti 3x3 ve formátu XML

3.19 Teorie o aplikaci matic na obraz

Dyadické operátory jsou v počítačové grafice velice využívané. Jsou používány na odstranění velkého množství vad obrazu. Základním principem dyadických operátorů je využití nejenom vstupního obrazu ale i matice aplikované na obraz. V našem digitálním světě je možné představit si obraz jako 2D matici a daný filtr taktéž jako 2D matici. Rozptylová matice bývá o velikosti maximálně velikosti obrazu. V praxi se používá mnohem menší matice, například 3x3 nebo 5x5. Matice mává lichou velikost v každém směru, to znamená, že není možné aplikovat matici o velikosti 2x2 či 4x4 atd. Není nutná čtvercová matice, ale bývá to standardem.

-1	-1	-1
-1	8	-1
-1	-1	-1

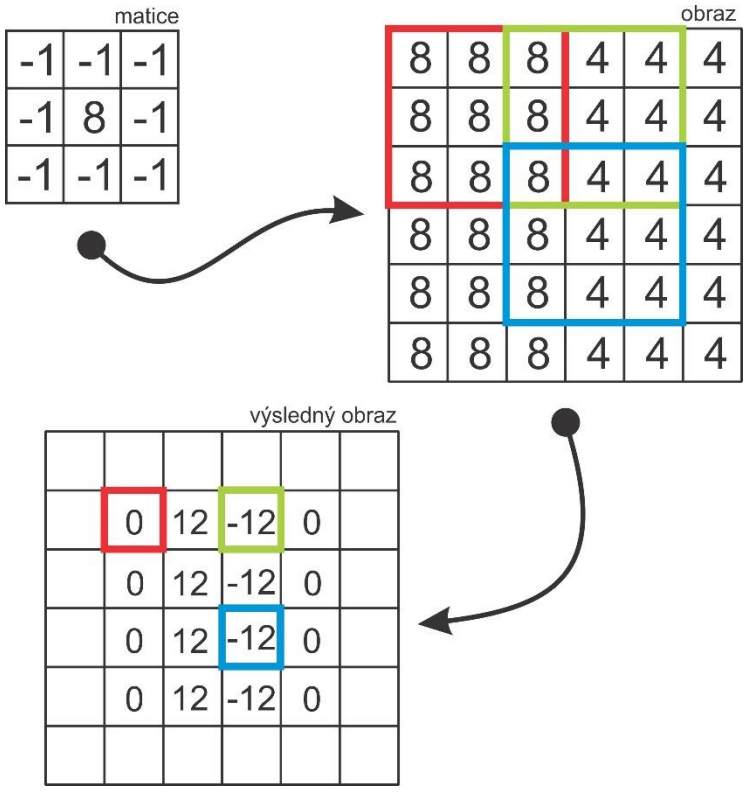
Obrázek 13 Ukázka matice pro dyadického operátoru na obraz

8	8	8	4	4	4
8	8	8	4	4	4
8	8	8	4	4	4
8	8	8	4	4	4
8	8	8	4	4	4
8	8	8	4	4	4

Obrázek 14 Ukázka obrazu pro aplikaci dyadického operátoru na obraz

Představme si obraz a matici jako desky, které je možné přikládat na sebe. Základním principem pro určení hodnoty daného pixelu nového obrazu na místě, je přiložení dané matice tak, aby střední pixel na matici byl položen na daném pixelu obrazu, který chceme určit. Vznikne nám překryv. Za pomoci tohoto překryvu vidíme, kolikrát máme který pixel z okolí daného pixelu přičíst nebo odečíst. Takto spočítáme hodnoty a zjistíme celkovou hodnotu pixelu v novém obrazu. Tento postup zopakujeme pro každý pixel původního obrazu. Hodnoty spočítaných pixelů zapisujeme do nového obrazu a vždy počítáme s původním obrazem.

Na obrázku níže je možné vypočítat hodnoty ve výsledném obrazu tak, že přiložíme matice na obraz a spočítáme roznásobené koeficienty matice s hodnotami z obrázku.



Obrázek 15 Ukázka dyadického operátoru

Pro **červený** pixel ve výsledném obrázku:
(-1)x8 + (-1)x8 + (-1)x8 + (-1)x8 + 8x8 + (-1)x8 + (-1)x8 + (-1)x8 + (-1)x8 = **0**

Pro **zelený** a **modrý** pixel ve výsledném obrázku:
(-1)x8 + (-1)x4 + (-1)x4 + (-1)x8 + 8x4 + (-1)x4 + (-1)x8 + (-1)x4 + (-1)x4 = **-12**

Poznámka: Pokud bude součet koeficientů v matici roven hodnotě 1, tak obraz bude zachovávat intenzitu. Jinak řečeno nebude po aplikaci filtru světlejší ani tmavší. Pokud bude součet koeficientů větší než 1, tak bude obraz světlejší a pokud bude menší než 1, tak bude obraz tmavší.

3.19.1 Rozmazání obrazu

Rozmazání obrazu se často používá pro odstranění vad obrazu jako je šum. Jedná se v podstatě o zprůměrování hodnot v malém okolí. Vzhledem na to, že matice je uvedena ve tvaru, kde se nachází zlomky a aritmetika v pohyblivé desetinné čárce je pomalejší než celočíselná, používá se matice. Tato matice je vyplněna jedničkami a až před zápisem do nového obrazu se výsledná hodnota dělí číslem v zlomku. V našem případě tedy číslem 9.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Obrázek 16 Matice filtru rozmazání

3.19.2 Zostření obrazu

Poměrně často bývá obraz rozostřen. Aplikací následného filtru se mění obraz tak, že se klade větší důraz na střední pixel a ostatní se od něho odpočítávají. Je důležité, aby součet hodnot matice byl roven 1.

Poznámka: Užívá se zde znalost derivací. Pro více informací o derivacích v digitálním obraze je možné navštívit stránku http://cs.wikipedia.org/wiki/Detekce_hran

-1	-1	-1
-1	9	-1
-1	-1	-1

Obrázek 17 Matice filtru zostření

3.19.3 Detekce hran v obrazu

Detekce hran je často užita pro automatickou detekci čísel, poznávacích značek, či detekci textu pro zrakově postižené osoby, které potřebují z obrázku přečíst informace. Využívá se také pro automatické rozpoznávání textu, tzv. OCR (http://cs.wikipedia.org/wiki/OCR) - optické rozpoznávání znaků.

-1	-1	-1
-1	8	-1
-1	-1	-1

Obrázek 18 Matice filtru detekce hran

Poznámka: Existuje veliké množství variací a adaptací tohoto filtru. Jeden z nejznámějších je Robertsův či Sobelův filtr.

3.19.4 Příklad k procvičení

Zadání: Doplňte hodnoty do výsledného obrazu. Hodnoty X nepočítejte.

1	**-2**	**1**
1	**0**	**1**
1	**-2**	**1**

Obrázek 19 Matice pro procvičení výpočtu dyadického operátoru

8	4	2	1	0
4	8	4	2	1
2	4	6	1	0
1	2	1	4	1
0	1	0	1	2

Obrázek 20 Obraz pro procvičení výpočtu dyadického operátoru

X	X	X	X	X
X	10			X
X				X
X				X
X	X	X	X	X

Obrázek 21 Výsledný obraz pro procvičení výpočtu dyadického operátoru

Ukázka řešení:

1***8** + **(-2)*****4** + **1*****2** + **1*****4** + **0*****8** + **1*****4** + **1*****2** + **(-2)*****4** + **1*****6** = 8 - 8 + 2 + 4 + 0 + 4 + 2 - 8 + 6 = **10**

3.20 Filtr pro konvoluci

Studenti v této části přímo užijí vědomostí z hodiny o aplikací matic na obraz. Implementace třídy `MatrixFilter` je přímočará, ale je nutné dávat si pozor na to, aby každý student vypracoval tento algoritmus a konkrétně metodu `applyFilter` samostatně. Pro studenty a jejich abstraktní uvažování se jedná o poměrně složitý algoritmus. Čtyřnásobný `for` cyklus se pro některé studenty stává téměř nepřekonatelný problém, no jiní ho zvládají během několika minut.

Modifikované třídy

`MatrixFilter, MainFrame`

3.20.1 MatrixFilter

Metoda `applyFilter` aplikuje matici na obraz. Algoritmus zde uveden je velice přímočarý a nevyužívá žádné urychlovací metody. Složitost algoritmu je $O(n^2)$. Pokud by bylo zapotřebí implementovat rychlejší algoritmus, je například možné použít jednu z technik uvedených v (Karas, Pavel; Svoboda, David;, 2013).

Vzhledem k tomu, že výsledný obraz je stejně velký jako vstupní, algoritmus musí řešit jak se zachovat pokud matice k výpočtu vyžaduje bod obrazu, který je mimo obraz samotný. V našem případě to řešíme tím, že vezmeme hodnotu nejbližšího pixelu v obrazu. Je to dosaženo za pomoci toho, že jakýkoliv index pro pixel, který je pod hodnotou 0 nahradíme hodnotou 0 a jakýkoliv index pro který je za šířkou resp. výškou nahradíme indexem krajního bodu na výšce resp. šířce.

Algoritmus prochází postupně obraz a v něm vždy celou matici. Vzhledem k tomu, že v matici mohou být zapsána pouze celá čísla je zde uveden koeficient a jestli je matice normalizována. Pokud je matice normalizována, je celková hodnota daného pixelu před uložením vynásobena hodnotou koeficientu. Taktéž pokud je hodnota daného pixelu v jakékoliv složce mimo limit, je hodnota upravena tak, aby se vešla mezi 0 až 255.

```
public void applyFilter() throws FilterException{
  if (originalImage == null || matrix == null){
    throw new FilterException();
  }
  filteredImage = new BufferedImage(originalImage.getWidth(),
                                    originalImage.getHeight(),
                                    originalImage.getType());
  for (int x = 0; x < originalImage.getWidth(); x++){
    for (int y = 0; y < originalImage.getHeight(); y++){
      int r = 0;
      int g = 0;
      int b = 0;
      for (int mX = 0; mX < matrix.length(0);mX++){
        for (int mY = 0; mY < matrix.length(1);mY++){
          if (matrix.get(mX,mY) != 0){
            int origXget = x+mX-matrix.length(0)/2;
            int origYget = y+mY-matrix.length(1)/2;
            if (origXget < 0) origXget = 0;
            if (origYget < 0) origYget = 0;
            if (origXget > originalImage.getWidth()-1)
              origXget = originalImage.getWidth()-1;
            if (origYget > originalImage.getHeight() -1)
              origYget = originalImage.getHeight() -1;
            Color c = new Color( originalImage.getRGB(origXget, origYget), false );
            r += c.getRed()*matrix.get(mX,mY);
            g += c.getGreen()*matrix.get(mX,mY);
            b += c.getBlue()*matrix.get(mX,mY);
          }
        }
      }
      if (matrix.normalised()){
        r = Math.round(r*matrix.getCoefficient());
```

```
          g = Math.round(g*matrix.getCoefficient());
          b = Math.round(b*matrix.getCoefficient());
        }
        if(r > 255) r = 255;
        if(g > 255) g = 255;
        if(b > 255) b = 255;
        if(r < 0) r = 0;
        if(g < 0) g = 0;
        if(b < 0) b = 0;
        Color col = new Color(r,g,b);
        filteredImage.setRGB(x,y,col.getRGB());
      }
    }
}
```

Kód 34 Metoda applyFilter třídy MatrixFilter

Obrázek 22 Ukázka výstupu metody applyFilter třídy MatrixFilter s různými maticemi

3.20.2 MainFrame

Úprava třídy `MainFrame` spočívá v tom, že po stlačení tlačítka *„Apply Matrix Filter“* se nastaví `MatrixFilter` a spustí se výpočet. Po proběhnutí výpočtu se překreslí obrázek.

3.21 Testování aplikace na rozšířené sadě testovacích příkladů.

Aplikaci v současném stavu chybí dialogové okno s popisem aplikace, některé prvky aplikace se ještě nezapínají a nevypínají tak, jak by se měli a dovolují uživatelovi mačkat tlačítka, které nemohou fungovat. Mimo jiné někteří žáci ještě mohou bojovat s implementací `XMLMatrixHandler` nebo `MatrixFilter`. Žákům je rozdána další sada JUnit testů pro jejich aplikaci.

Dodané třídy

`AboutDialogTester, XMLHandlerTester, MatrixFilterTester`

Modifikované třídy

`AboutDialog`

3.22 Kontrolní den

Odevzdané úkoly pedagog zkontroluje a identifikuje nejčastější chyby, které se v pracích nacházely. Tyto chyby následně analyzuje před žáky a osvětlí, proč jsou dané postupy zvolené nevhodně nebo naopak ukáže řešení, které tyto chyby obchází.

3.23 Pixelizační filtr

Filtr `PixelizerFilter` již nebude dále testován a slouží jako pracovní výplň pro žáky, který jsou moc napřed a mají již mnoho vypracováno a na hodinách se nudí. Je to grafický filtr a jeho implementace není jasně daná. Jde o to, aby žáci svými vlastními silami uvažovali o tom, jak daný filtr vytvořit. Algoritmus schovaný za jednou z jednodušších implementací je pro některé studenty i tak poměrně netriviální.

Modifikované třídy

`PixelizerFilter`

3.23.1 PixelizerFilter

Algoritmus pro vytvoření rozpixlovaného efektu na obrazu je postaven na principu vzorkování a uložení této vzorky do všech pixelů v okolí. Velikost okolí je specifikováno uživatelem, který může posouvat instancí `JSlider` a tak určovat velikost tohoto okolí. Algoritmus teda vzorkuje s frekvencí `factor*2` a všude v okolí zapisuje hodnotu pixelu určeného tímto vzorkem. Kontrola čtení mimo obraz je zabezpečena metodami `Min` a `Max`, které limitují výběr.

```
public void applyFilter(BufferedImage image) throws FilterException{
   try {
      filteredImage = new BufferedImage(image.getWidth(), image.getHeight(),
                                        image.getType());
      for (int x = 0; x < image.getWidth();x = x + factor * 2){
         for (int y = 0; y < image.getHeight();y = y + factor * 2){
            for (int a = x - factor; a < x + factor; a++){
               for (int b = y - factor; b < y + factor; b++){
                  int getA = Math.min(image.getWidth()-1, Math.max(0, a));
                  int getB = Math.min(image.getHeight()-1, Math.max(0, b));
                  Color c = new Color(image.getRGB(x, y));
                  filteredImage.setRGB(getA, getB, c.getRGB());
               }
            }
         }
      }
      ((CustomJPanel)customJPanel).setImage(filteredImage);
      ((CustomJPanel)customJPanel).repaint();
   } catch (Exception e) {
      throw new FilterException();
   }
}
```

Obrázek 23 Metoda applyFilter třídy PixelizerFilter

Obrázek 24 Ukázka aplikace pixelizačního filtru na obraz

3.24 Implementace logování aplikace

Implementace vlastního logování v aplikaci je zredukována pouze na výpis prováděné operace do instance třídy `JTextArea`. Nechte, ať studenti samostatně vypracují logování a rozhodnou se, co všechno je nutné logovat.

Modifikované třídy

`MainFrame`

3.25 Návrh vlastního filtru obrazu a implementace

Návrh vlastního filtru je povinná úloha pro studenty. V této době by měli mít dostatečné schopnosti pro vytvoření vlastního filtru. Tento filtr musí mít vlastní GUI interface a jeho proměnné musí být nastavitelné. Filtr musí implementovat interface `ImageFilter`.

4 Závěr

Projekt je postaven tak, aby studenti na konci celého ročníku měli funkční aplikaci. Postupným zvětšováním aplikace studenti chápou složitost projektu. Za pomoci teoretické látky pochopí problematiku a později se naučí převést tyto vědomosti do praktické podoby. Pokud pedagog bude postupovat podle stanoveného plánu, který je nastíněn, je možné projekt dovést do zdatného konce.

Projekt je poměrně náročný na čas a je nutné předpokládat, že někteří studenti budou muset doma dodělávat práci z hodiny. Přesně pro tento účel mají dostupnou dokumentaci k projektu. Osobně se mi tento přístup vyplatil a jeden ročník studentů už podle tohoto projektu fungoval. V době běhu tohoto projektu jsem nezaznamenal žádné výraznější problémy.

5 Seznamy

5.1 Seznam obrázků

5.2 Seznam zdrojových kódů

5.3 Seznam Vzorců

6 Použité zdroje

Karas, Pavel; Svoboda, David;. 2013. Algorithms for Efficient Computation of Convolution. *Design and Architectures for Digital Signal Processing.* místo neznámé : InTech, 2013, 8, str. 322.

Žára, Jiří, Beneš, Bedřich a Sochor, Jiří. 2004. *Moderní počítačivá grafika.* Brno : Computer Press, 2004. ISBN: 80-251-0454-0.