



Zpracování multimediálních dat – cvičení

Garant předmětu:
doc. Ing. Petr Číka, Ph.D.

Autoři textu:
doc. Ing. Petr Číka, Ph.D.
Ing. David Kohout

BRNO 2023

Autor doc. Ing. Petr Číka, Ph.D., Ing. David Kohout

Název Zpracování multimediálních dat – cvičení

Vydavatel Vysoké učení technické v Brně
Fakulta elektrotechniky a komunikačních technologií
Ústav telekomunikací
Technická 12, 616 00 Brno

Vydání první

Rok vydání 2023

Verze 3

Náklad elektronicky

Tato publikace neprošla redakční ani jazykovou úpravou

Obsah

0 Úvod	3
0.1 Instalace vývojového prostředí knihoven	3
0.2 Základy práce s vývojovým prostředím IntelliJ IDEA	3
0.3 Instalace SceneBuilder	3
1 Cvičení 1 – Úvod do JavaFX	4
1.1 Vytvoření projektu	4
1.2 Tvorba rozhraní	5
1.3 Spuštění hlavního okna	6
2 Cvičení 2 – Převody barevných modelů	7
2.1 Teoretická část	8
2.1.1 Barevný model <i>RGB</i>	8
2.1.2 Barevný model <i>YCbCr</i>	9
2.2 Praktická část	10
2.2.1 Načtení a zobrazení digitálního obrazu	10
2.2.2 Metoda pro získání matic <i>RGB</i> z načteného obrazu	11
2.2.3 Metoda pro vytvoření nového obrazu z matic <i>RGB</i>	12
2.2.4 Zobrazení pouze jedné barevné složky	12
2.2.5 Ověření funkčnosti implementovaných metod	13
2.3 Samostatná práce	13
2.3.1 Na co si dát pozor?	14
2.4 Jak otestovat?	14
2.5 Bonus	15
2.6 Jak odevzdat?	15
3 Cvičení 3 – Vzorkování	16
3.1 Cíle cvičení	16
3.2 Teoretická část	16
3.3 Praktická realizace	18
3.3.1 Zobrazení komponent barevného modelu <i>RGB</i>	18
3.3.2 Podvzorkování modelu <i>YCbCr</i>	19
3.3.3 Nadvzorkování modelu <i>YCbCr</i>	22
4 Cvičení 4 – Objektivní hodnocení kvality	24
4.1 Cíle cvičení	24
4.2 Teoretická část	24
4.2.1 Metody pro měření kvality obrazových dat	24
4.2.2 Subjektivní měření kvality	24
4.2.3 Objektivní měření kvality	25
4.3 Praktická realizace	28
4.3.1 Implementace algoritmů kvality	28
4.3.2 Úprava grafického rozhraní	28

5 Cvičení 5 – Transformace	31
5.1 Cíle cvičení	31
5.2 Teoretická část	31
5.2.1 Transformační kódování	31
5.3 Praktická realizace	37
5.3.1 Vytvoření třídy a kostry metod	37
5.3.2 Implementace metod pro vytvoření transformačních matic libovolných rozměrů	37
5.3.3 Implementace obecné metody pro 2D transformaci	38
5.3.4 Propojení se zbytkem aplikace	38
5.3.5 Provedení transformace rozdělením obrazu na bloky velikosti $N \times N$	38

0 Úvod

Učební text slouží k podpoře cvičení předmětu Zpracování multimediálních dat. Průběh cvičení je rozdělen do dvou částí:

1. V první části semestru bude úvodní cvičení, kde se vytvoří rozhraní aplikace pomocí JavaFX. Následně bude následovat 5 cvičení zaměřených na implementaci algoritmů využívaných při kompresi obrazu a videa technikami JPEG, MPEG, H.26x.
2. Celý semestr - práce na individuálním projektu, od šestého cvičení budou probíhat pouze konzultace k řešení projektů.

0.1 Instalace vývojového prostředí knihoven

1. Nainstalujte Java SE Development Kit v aktuální verzi dostupné [na webových stránkách www.oracle.com](#).
2. Nainstalujte aktuální verzi IntelliJ IDEA Community dostupnou [na stránkách www.jetbrains.com](#).

0.2 Základy práce s vývojovým prostředím IntelliJ IDEA

- Prostudujte základní rozložení a ovládání vývojového prostředí na <https://www.jetbrains.com/help/idea/>
- Naučte se, jak vytvořit aplikaci JAVA - kapitola Create your first application. Naučte se vytvořit spustitelný JAR soubor.
- Naučte se používat klávesové zkratky - Keyboard shortcuts, což Vám velmi pomůže při dalším vývoji aplikací.
- V nastavení (CTRL+ALT+S) je vhodné vyhledat **Code completion** a zde zrušit zaškrtnutí políčka **Match case**.

0.3 Instalace SceneBuilder

Pro vytvoření rozhraní aplikace budeme používat framework JavaFX. Pro usnadnění tvorby rozhraní slouží aplikace SceneBuilder. Tu stáhneme ze stránek <https://gluonhq.com>. Po nainstalování je možné SceneBuilder nalinkovat přímo do IntelliJ, pak je možné rozhraní vytvářet přímo uvnitř IntelliJ, nebo lze jednoduše SceneBuilder spustit. V IntelliJ můžete nainstalovat plugin JavaFX (měl by ale být součástí) a SceneBuilder nalinkovat v nastavení a vyhledat JavaFX (záložka Languages & Frameworks -> JavaFX), zde se vloží cesta k SceneBuilder.exe. Takto je možné otevřít FXML soubory kliknutím pravým tlačítkem a zvolit položku Open In SceneBuilder (2. od konce). Alternativně lze otevřít FXML soubor přímo a v dolní části okna přepnout zobrazení z Text na Scene Builder. Poslední možností je SceneBuilder spustit manuálně a potřebný FXML soubor vyhledat a otevřít z průzkumníku.

1 Cvičení 1 – Úvod do JavaFX

1.1 Vytvoření projektu

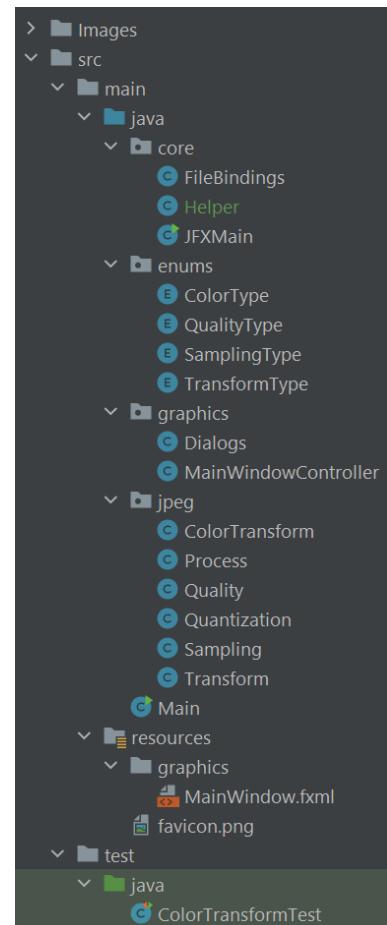
V IntelliJ IDEA si založte nový projekt. Doporučený postup: **File -> New -> Project**. V novém okně stačí vybrat nejzákladnější možnost, tedy **New Project** a zde si nastavit název, umístění, verzi Javy (**Java 11** je v učebně). Poslední je **Build system** ten nastavte na **Maven**.

Projekt je vhodné rozdělit do několika logických celků (balíčků). Ve složce `java` (složka s kódem) je vhodné použít následujících balíčků (New -> Package): Core, Enums, Graphics a Jpeg. Ve složce `resources` je dále potřebná složka se stejným názvem, kde se bude nacházet soubor s rozhraním, tedy složka `Graphics`. Celá struktura projektu, včetně základních tříd je na obrázku 1.1. Tato struktura odpovídá vzorovému programu po všech cvičeních.

Od Javy 9 již není JavaFX součástí oficiálních balíčků Javy, proto je nutné si JavaFX stáhnout jako dodatečnou knihovnu. K tomu nám slouží právě Maven, který jsme si zvolili při tvorbě projektu. Nyní je nutné rozšířit soubor `pom.xml` o pár řádků. Do elementu `<project>` před jeho uzavřením vložíme nový element `<dependencies>` a dovnitř vložíme potřebné knihovny:

```
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>16</version>
</dependency>

<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>16</version>
</dependency>
```

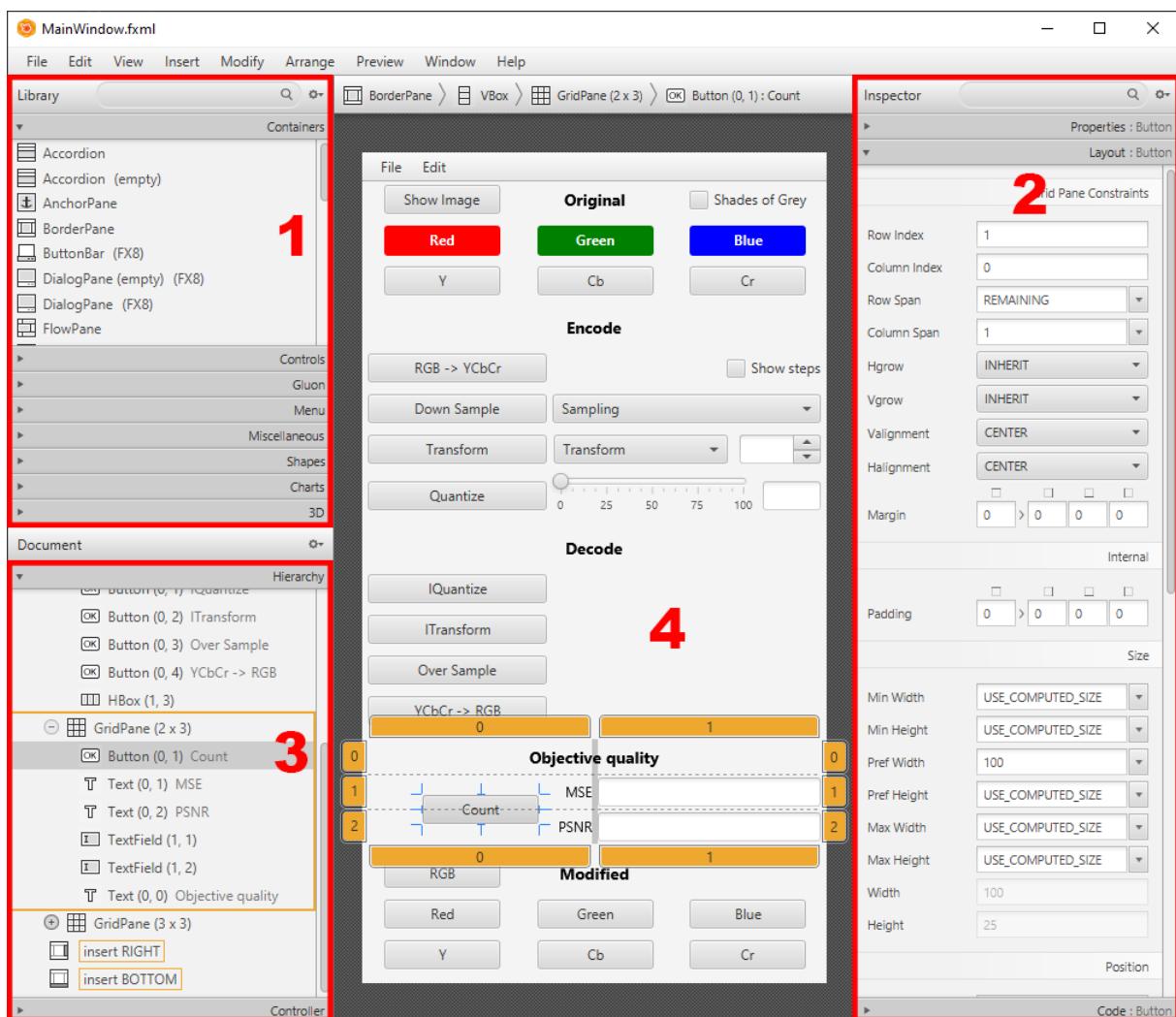


Obrázek 1.1: Struktura

1.2 Tvorba rozhraní

Tvorba rozhraní přes SceneBuilder je velice jednoduchá.

Obrázek 1.2 představuje ovládací prvky aplikace. V prohlížeči komponentů {1} je možné nalézt potřebný ovládací prvek, který se následně přesune do pracovní oblasti, kde budeme celé rozhraní aplikace {4}. Každý vložený prvek lze upravit v pravém podokně {2}. Tato sekce je rozdělena do 3 částí. První (**Properties**) upravuje základní nastavení, druhá (**Layout**) nastavení spojené s umístěním a rozložením v okně. Poslední sekce (**Code**) řeší propojení s kódem, nejdůležitější položky v této sekci jsou `fx:id` tím můžeme propojit prvek z proměnou v kódu a druhou položkou je **On Action**, která slouží pro nastavení názvu metody, která se provede po např. stisknutí tlačítka.



Obrázek 1.2: Scene Builder

Část {3} představuje hierarchické zobrazení celého rozhraní. Je zde možné vybírat i elementy, které nejsou viditelné v grafickém znázornění. Také je zde možné upravit pořadí prvků (pozn: pořadí prvků souvisí například s ptoklikáním se po elementech tab při spuštěné aplikaci.) Taktéž se v této části nastavuje Controller, který bude dané okno používat jako propojení s kódem.

Vytvořené rozhraní je možné zobrazit klávesovou zkratkou CTRL+P, případně přes menu Preview -> Show Preview in Window. Poslední, co stojí za zmínku je automatické vytvoření kostry kontrolní třídy, které je možné provést přes View -> Show Sample Controller Skeleton. (**Celé grafické rozhraní je vhodné vytvořit celé najednou a připravit veškeré ID a volání metod. Až je celé okno připravené, teprve poté je vhodné si vygenerovat Controller a vložit jej do kódu.**)

1.3 Spuštění hlavního okna

Pro spuštění aplikace je nutné vytvořit třídu, která provede inicializaci JavaFX, načež vytvořené rozhraní a zobrazí jej do okna. Výsledná třídě je znázorněná na obrázku 1.3. Tato třída rozšiřuje třídu Application. Kód uvnitř metody start je v podstatě minimální potřebný kód pro spuštění aplikace. Není nutné nastavovat titulek, ikonu a akci při ukončení. **Aplikace musí být spuštěna metodou main, která obsahuje volání launch.** Bohužel od Javy 9 nejde tento main spustit napřímo, a tak je nutné vytvořit další spustitelnou třídu, kde se v metodě main zavolá main této třídy: `JFXMain.main(args);`.

```
public class JFXMain extends Application { 2 usages
    private static Stage primaryStage; 6 usages
    public static Scene mainScene; 2 usages

    @Override
    public void start(Stage stage) throws Exception {
        primaryStage = stage;

        FXMLLoader fl = new FXMLLoader(FilePaths.GUIMain);
        Parent root = fl.load();
        mainScene = new Scene(root);
        primaryStage.setScene(mainScene);

        primaryStage.setTitle("JPEG: Příjmení VUTID"); //Titulek okna, nastavte svoje
        primaryStage.getIcons().add(FilePaths.favicon); //Přidání ikony aplikace
        primaryStage.show(); //Zobrazí rozhraní

        //Není nutné, umožňuje provést kód po stisku X (př. potvrzení ukončení)
        primaryStage.setOnCloseRequest((e) -> {
            Platform.exit();
            System.exit(status: 0);
        });
    }

    //Nutné mít launch uvnitř main metody.
    //Bohužel nejde spustit přímo z této třídy.
    public static void main(String[] args) { 1 usage
        launch(args);
    }
}
```

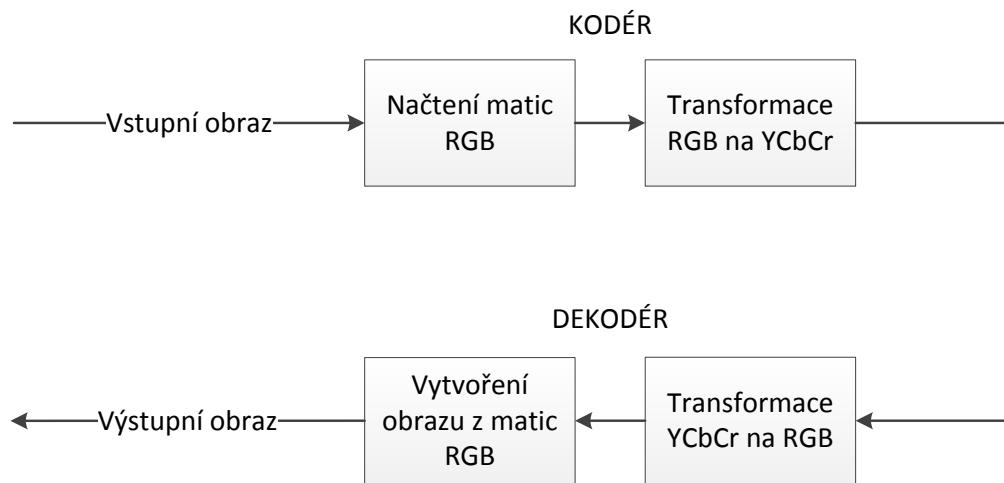
Obrázek 1.3: JFXMain

2 Cvičení 2 – Převody barevných modelů

V předchozím cvičení jsme si připravili základní strukturu projektu. V tomto cvičení se naváže na tento základ. Prostudujte si teoretickou část tohoto cvičení, kde se naučíte základní práci s digitálními obrazy. Během cvičení bude vytvořena kostra programu pro demonstraci technik využívaných v kompresním standardu JPEG. Vytvořený program bude podporovat následující funkce:

1. Načtení digitálního obrazu z předem zvoleného souboru.
2. Zobrazení načteného digitálního obrazu.
3. Získání matic R, G, B z digitálního obrazu.
4. Transformace z prostoru RGB do $YCbCr$ a naopak.
5. Vytvoření digitálního obrazu z matic RGB .

Popsané funkce jsou graficky znázorněny na obrázku 2.1.



Obrázek 2.1: Schéma kodéru i dekodéru

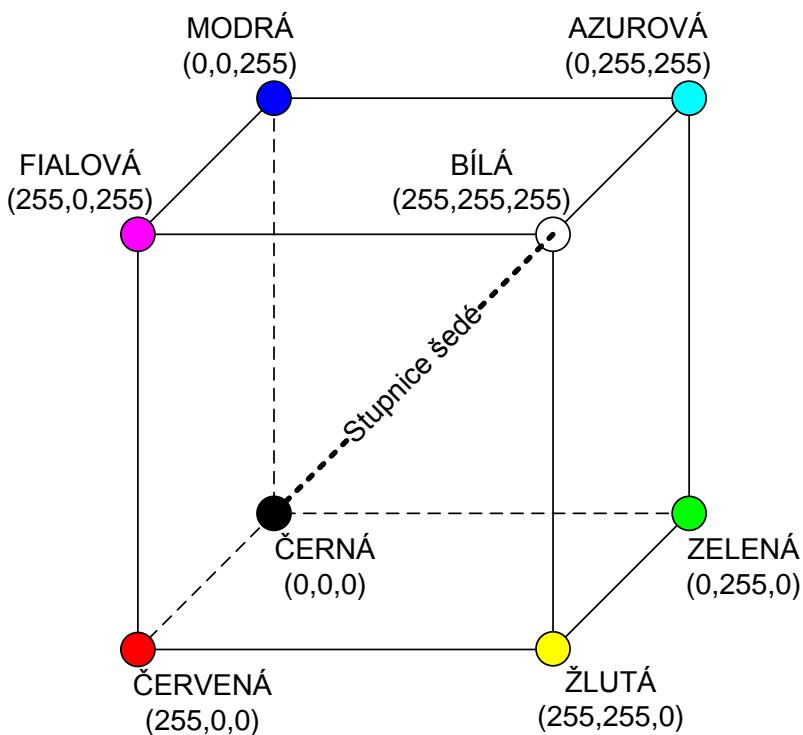
2.1 Teoretická část

V teoretické části bude rozebrán barevný model *RGB* a *YCbCr* a vzájemné vztahy mezi nimi.

2.1.1 Barevný model *RGB*

Barevný model *RGB* využívá aditivní míchání barev, konkrétně červené *R*(*Red*), zelené *G*(*Green*) a modré *B*(*Blue*). Aditivní míchání znamená, že sečtením jednotlivých barevných složek vznikne výsledná barva. Barevný model *RGB* lze vyjádřit jednotkovou krychlí – viz obr. 2.2.

Variantou modelu *RGB* je model *ARGB*, kde je ke třem základním barvám přidán alfa kanál definující průhlednost snímku.



Obrázek 2.2: Barevný model *RGB*

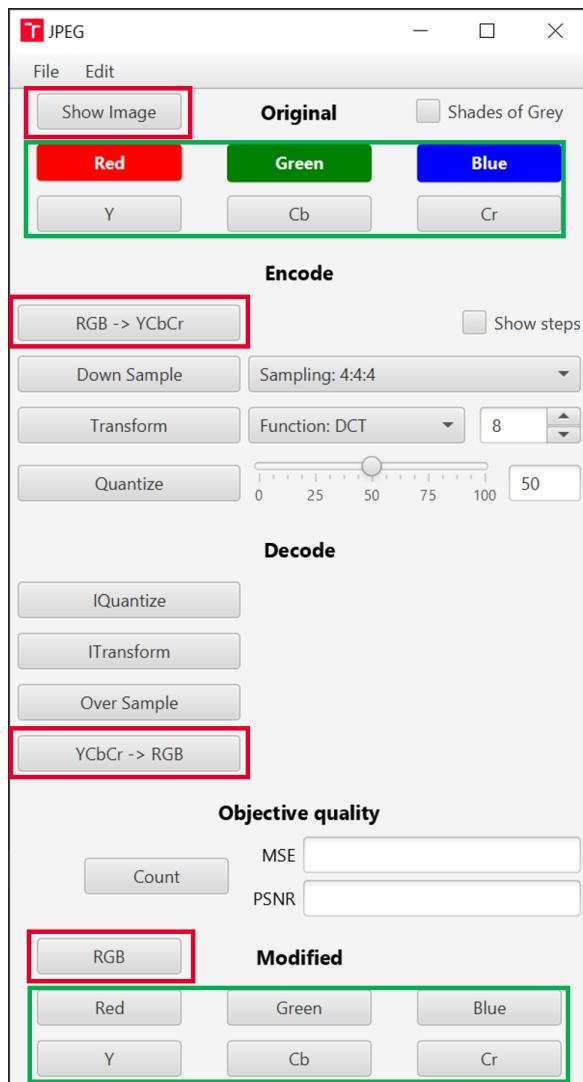
Model *RGB* je používán ve všech barevných zobrazovačích (televize, displeje apod.). Model není vhodný použít při kompresích standardy JPEG, MPEG apod. vzhledem k tomu, že při odstranění jakékoli informace lidské oko ihned rozpozná změnu obrazu. Z toho důvodu se při kompresích využívají modely oddělující jasovou a barvonosnou složku, jako je např. model *YUV* nebo *YCbCr*. V následující podkapitole bude podrobně rozebrán model *YCbCr* vzhledem, který je používán při zpracování digitálními kompresními standardy JPEG, MPEG a H.26x.

2.1.2 Barevný model $YCbCr$

Barevný model $YCbCr$ je používán v kompresních standardech JPEG, MPEG, H.26x vychází z následujících transformačních rovnic:

$$\begin{aligned} Y &= 0,257R + 0,504G + 0,098B + 16, \\ Cb &= -0,148R - 0,291G + 0,439B + 128, \\ Cr &= 0,439R - 0,368G - 0,071B + 128, \end{aligned} \quad (2.1)$$

$$\begin{aligned} R &= 1,164(Y - 16) + 1,596(Cr - 128), \\ G &= 1,164(Y - 16) - 0,813(Cr - 128) - 0,391(Cb - 128), \\ B &= 1,164(Y - 16) + 2,018(Cb - 128). \end{aligned} \quad (2.2)$$



Obrázek 2.3: Náplň práce ve druhém cvičení

2.2 Praktická část

Ve vývojovém prostředí IntelliJ IDEA pokračujte v projektu z předešlého cvičení a případně si upravte základní JavaFX rozhraní. Dále proveďte následující kroky:

- Nainstalujte knihovnu JAMA Matrix v souboru pom.xml pomocí těchto řádků:

```
<dependency>
    <groupId>gov.nist.math</groupId>
    <artifactId>jama</artifactId>
    <version>1.0.3</version>
</dependency>
```

- Ve složce src uvnitř složky java vytvořte balíček (package) s názvem jpeg, uvnitř kterého budou ve výsledku implementovány všechny potřebné třídy a metody.
- V balíčku jpeg vytvořte prázdné třídy Process.java a ColorTransform.java.
- Pokud ještě nemáte, tak si z e-learningu si stáhněte třídu Dialogs.java a vložte ji do balíčku graphics. Tato třída implementuje některé základní funkce, které budete potřebovat pro načtení obrázků a jejich zobrazení.
- V adresáři projektu (ve stejně složce jako je složka src) vytvořte složku Images a sem nahrajte originální obraz dostupný z e-learningu Lenna.png

Celá struktura po všech krocích bude podobná struktuře projektu jako na obrázku 1.1. Před jakoukoliv prací je vhodné si společně definovat pořadí výšky a šířky v 2D polích a maticích. Knihovna JAMA Matrix, kterou budete používat definuje new Matrix(rows, columns), od toho si určíme, že první bude vždy výška (height neboli Y souřadnice) a druhý parametr bude šířka (width neboli X souřadnice). Toto je důležité především u barevných polí, např: int [][] red = new int [height][width].

2.2.1 Načtení a zobrazení digitálního obrazu

Pro načtení digitálního obrazu existuje v jazyku Java mnoho způsobů. Pro naše účely můžete využít předpřipravenou třídu Dialogs.java, která implementuje načtení obrázku (BufferedImage) z cesty, dále umožňuje vyvolat Open File dialog a nakonec umí zobrazit obrázek v okně aplikace. Ve třídě FileBindings vytvořte cestu k výchozímu obrázku (např. String defaultImage = "Images/Lenna.png"). S touto proměnnou následně pracujte. Obrázek z cesty můžete načíst pomocí metody Dialogs.loadImageFromPath(cesta), to vám vrátí BufferedImage. Následně můžete tento obrázek zobrazit pomocí metody Dialogs.showImageInWindow(image, title), to vám zobrazí obrázek v novém okně aplikace. MainWindowController bude obsahovat **globální proměnnou na objekt třídy Process**, bude tak možné jednoduše objekt načíst znovu s jiným obrázkem, případně obnovit stav při resetování postupu. Při inicializaci okna si inicializujte objekt Process, kde do konstruktoru vstoupí výchozí obrázek, definovaný dříve. **Třída MainWindowController bude pouze sloužit k ovládání aplikace a bude volat metody uvnitř objektu Process, právě třída Process bude řídit logiku aplikace (kodéru).**

Deklarace potřebných polí

Uvnitř třídy Process si vytvořte několik proměnných (pro usnadnění mohou být všechny public, pokud si nechcete vytvářet gettery a settery, ale private je běžná konvence). První proměnnou bude originální obrázek BufferedImage a proměnné pro výšku a šířku (načtete z originálního obrázku):

- `private BufferedImage originalImage;`
- `private int imageHeight;`
- `private int imageWidth;`

Jak je patrné z teoretické části, obraz se skládá ze tří matic *R*, *G*, *B* (červená, zelená, modrá). Standardní digitální obraz je vyjádřen 24 bity na jeden pixel, tzn. pro jednu barvu je to 8 bitů. V nově vytvořené třídě si připravíme 3 privátní (pro usnadnění veřejné) dvouozměrná pole typu `int` a nazveme je například `red`, `green`, `blue`. Je vhodné připravit si rovnou 2 varianty těchto polí pro originální a modifikovaná data. Pro deklaraci polí použijte následující:

- `private int [][] originalRed, modifiedRed;`
- `private int [][] originalGreen, modifiedGreen;`
- `private int [][] originalBlue, modifiedBlue;`

Dále deklarujeme tři privátní matice `y`, `cB`, `cR` typu `Matrix`, opět duplicitně pro originální a modifikovaná data:

- `private Matrix originalY, modifiedY;`
- `private Matrix originalCb, modifiedCb;`
- `private Matrix originalCr, modifiedCr;`

Modifikované verze matic najdou uplatnění především v dalších cvičeních, kde nám umožní zobrazit průběh celého kodéru a kontrolovat, zda dané modifikace skutečně fungují.

2.2.2 Metoda pro získání matic *RGB* z načteného obrazu

Z proměnné typu `BufferedImage` lze získat barevnou hodnotu `Color` daného pixelu pomocí následujícího volání: (`x` a `y` definují pozici daného pixelu)

- `Color color = new Color(image.getRGB(x, y));`

Tato proměnná obsahuje barevnou informaci daného pixelu souhrnně, obsahuje tedy RGB hodnotu. Jednotlivé barevné složky je možné získat z této proměnné pomocí metod:

- `getRed()`,
- `getGreen()`,
- `getBlue()`,

Pro naplnění matic `red`, `green`, `blue` můžete použít `for` cyklus naznačený na obrázku 2.4, kde stejným způsobem doplňte ostatní matice `green`, `blue`.

```
private void setOriginalRGB() { 1 usage
    for (int h = 0; h < height; h++) {
        for (int w = 0; w < width; w++) {
            Color color = new Color(originalImage.getRGB( x: w, y: h));
            originalRed[h][w] = color.getRed();
            //doplnit green a blue
        }
    }
}
```

Obrázek 2.4: Metoda pro získání barevných složek RGB z obrázku

2.2.3 Metoda pro vytvoření nového obrazu z matic *RGB*

V předchozí kapitole jsme si ukázali, jak získáme hodnoty pixelů jednotlivých barevných matic obrazu. Tento proces je uplatněn v kodéru. V dekodéru naopak získáme barevné matice R , G , B a je z nich zapotřebí vytvořit finální barevný obraz. K tomu si vytvoříme novou metodu `getImageFromRGB`, která bude vracet proměnnou typu `BufferedImage`. Metoda je znázorněna na obrázku 2.5

```
public BufferedImage getImageFromRGB() { 1 usage
    BufferedImage bfImage = new BufferedImage(
        width, height,
        imageType: BufferedImage.TYPE_INT_RGB);

    for (int h = 0; h < height; h++) {
        for (int w = 0; w < width; w++) {
            bfImage.setRGB( x: w, y: h,
                (new Color( r: modifiedRed[h][w],
                            g: modifiedGreen[h][w],
                            b: modifiedBlue[h][w])).getRGB());
        }
    }
    return bfImage;
}
```

Obrázek 2.5: Metoda pro vytvoření obrazu z barevných složek modelu *RGB*

2.2.4 Zobrazení pouze jedné barevné složky

Metodou z předchozího bodu (2.2.3) je možné se inspirovat i pro metody pro zobrazení jednobarevných složek obrázku. Pouhý rozdíl je v použité barevné matici, a také v tom, do které barevné složky danou hodnotu vložíme (pokud vložíme hodnotu do všech 3 složek, dostaneme šedý obrázek). Zobrazení YCbCr složek je možné stejným způsobem, akorát je nutné hodnoty zaokrouhlit a oříznou hodnoty mimo rozsah 0-255. Doporučeným postupem pro tuto implementaci jsou 2 metody, které vracejí objekt typu `BufferedImage`:

- `showOneColorImageFromRGB(int [][] color, ColorType type, boolean greyScale)`
 - Parametr (Enum) `ColorType` určí, do které barevné složky se vloží daná barva.
 - Pokud chceme zobrazit složku v odstínech šedé, tak se hodnota musí vložit do všech 3 barevných složek.
- `showOneColorImageFromYCbCr(Matrix color)`
 - Hodnotu z matic složek (YCbCr) je nutné zaokrouhlit.
 - Pokud je hodnota < 0, nastavit na 0.
 - Pokud je hodnota > 255, nastavit na 255.
 - Hodnotu vkládáme do všech 3 barevných složek pro získání šedého obrázku (barevný u tohoto modelu nemá smysl).

Problémem u zobrazení YCbCr obrázku je to, že tento model není přímo určen k zobrazení. Hodnoty jsou ale v podobném rozsahu a tak je možné je aplikovat do RGB matic.

Výsledné zobrazení je ale pouze pro naše potřeby abychom viděli změny, které se aplikují při jednotlivých metodách.

2.2.5 Ověření funkčnosti implementovaných metod

Abychom ověřili, že implementované metody fungují dle předpokladů, provedeme jednoduchý test. Načteme obraz, který následně zobrazíme, získáme z něj matice RGB, z těchto matic vytvoříme nový obraz a ten opět zobrazíme. Oba digitální obrazy by měly být shodné.

Funkčnost implementujte do vytvořeného rozhraní. Tedy na tlačítko Show Image u originálního obrázku a Show RGB u modifikovaného. Využijte k tomu proměnné, které jste si vytvořili dříve original a modified red/green/blue. Po stisknutí tlačítka Show original image se zobrazí obrázek z uložené proměnné BufferedImage. Pro zobrazení modifikovaného RGB bude využívat metodu z obrázku 2.5) (jen bude potřeba naplnit matici R/G/B modifikované).

Nyní spusťte celý program. Po kliknutí na obě tlačítka by výsledkem měla být dvě okna zobrazující vizuálně stejný digitální obraz.

Celý proces můžete zkousit drobně upravit. Zkuste si prohodit 2 různé barevné složky, při finálním sestavení obrázku. Pouze pro otestování, zda vám vše funguje jak má. (**PS:** nezapomeňte následně upravit zpátky.)

2.3 Samostatná práce

Dokončete metody popsané v předchozí části. Poté navážte samostatnou prací. Výsledkem vaší práce by měla být funkční červeně zobrazená tlačítka na obrázku 2.3. Zeleně označená tlačítka je vhodné doimplementovat si do dalšího cvičení (jejich funkčnost není hodnocena), Ve třídě `ColorTransform.java` vytvořte dvě metody pro převod z barevného modelu *RGB* na *YCbCr* a naopak. Metody budou statické, tzn. že je bude možné volat z jiné třídy bez inicializace objektu. Vstupní parametry budou jednotlivé barevné složky. Návratové hodnoty budou vracet převedené matici. Metody definujte shodně jako na obrázku 2.6. (Shodná definice je důležitá, aby bylo možné testovacím souborem otestovat správnost vašich výpočtů). Pro implementaci zvolte přepočet pro SDTV signál, tzn. rovnice:

$$\begin{aligned} Y &= 0,257R + 0,504G + 0,098B + 16, \\ Cb &= -0,148R - 0,291G + 0,439B + 128, \\ Cr &= 0,439R - 0,368G - 0,071B + 128, \end{aligned} \tag{2.3}$$

pro převod *RGB* na *YCbCr* a rovnice

$$\begin{aligned} R &= 1.164(Y - 16) + 1,596(Cr - 128), \\ G &= 1,164(Y - 16) - 0,813(Cr - 128) - 0,391(Cb - 128), \\ B &= 1,164(Y - 16) + 2,018(Cb - 128). \end{aligned} \tag{2.4}$$

pro převod *YCbCr* na *RGB*.

Metody na obrázku 2.6 dodržte 1:1, aby bylo možné provést test pomocí Unit testů. První metoda vrací pole `Matrix` [], to představuje vrácení 3 složek modelu YCbCr. Podobně je tomu tak i u druhé metody při konverzi zpátky do `RGB`. Pole `int` [] [] [] by měl obsahovat 3 složky `RGB`, všechny typu `int` [] []. Jedná se o nejjednodušší způsob, jak může metoda vrátit více hodnot najednou. V obou případech jednotlivé hodnoty získáte z navrácených polí standardním způsobem (př: `result[0]`).

```
public static Matrix[] convertOriginalRGBtoYCbCr(int[][][] red, int[][][] green, int[][][] blue) {
    return new Matrix[]{convertedY, convertedCb, convertedCr};
}

public static int[][][][] convertModifiedYCbCrtoRGB(Matrix Y, Matrix Cb, Matrix Cr) {
    return new int[][][][]{convertedRed, convertedGreen, convertedBlue};
}
```

Obrázek 2.6: ColorTransfom metody

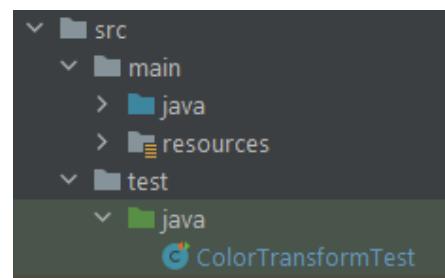
2.3.1 Na co si dát pozor?

1. Matice `y`, `cB`, `cR` jsou typu `Matrix`. Pro zápis hodnoty do matice použijte následující metodu:
 - `set(int i, int j, double s)`,
 kde `i`, `j` značí pozici daného pixelu v matici a `s` je hodnota pixelu.
2. Při zpětné konverzi z prostoru `YCbCr` do `RGB` může nastat přetečení rozsahu 0–255, kterého mohou dosahovat jednotlivé pixely. Proto při výpočtu použijte metodu `Math.round()` pro zaokrouhlení a poté ještě každou hodnotu testujte zvlášť a v případě:
 - (a) že hodnota > 255 – přiřaďte hodnotě 255,
 - (b) že hodnota < 0 – přiřaďte hodnotě 0.

2.4 Jak otestovat?

Do pom.xml si přidejte knihovnu pro testování kódu:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>RELEASE</version>
    <scope>test</scope>
</dependency>
```



Obrázek 2.7: Testovací složka

Do složky `src` vložte obsah zip souboru z e-learningu. Výsledkem bude to, že složka `src` bude obsahovat složky `main` a `test`.

Složka java uvnitř složky test by měla zezelenat jako na obrázku 2.7. Pokud se tak nestalo, klikněte pravým tlačítkem na projekt, zvolte Open Module Settings (funguje i klávesová zkratka F4). Nyní vidíte otevřenou záložku Modules, kde si vyberte složku java uvnitř složky test a klikněte nad výběrem souborů na zelenou složku Mark as: Tests.

Test spusťte stejně jako kód, jen pravým tlačítkem klikněte na třídu ColorTransformTest uvnitř složky test a java a zvolte Run.

Následně se provede otestování vašich metod uvnitř vaší třídy ColorTransform. Pokud bude vše v pořádku, dostanete 2 kladná hlášení:

- Test for conversion from RGB to YCbCr: OK
- Test for conversion from YCbCr to RGB: OK

Pokud bude nějaká část chybně, tak dojde k výpisu chybové hlášky, kde se chyba pravděpodobně nachází.

2.5 Bonus

V menu aplikace File -> Change Image, implementujte logiku, která dokáže změnit výchozí obrázek (například vytvořením nové instance objektu třídy Process). Kód pro výběr obrázků ze souborů je součástí třídy `Dialogs.java`. Dalším bonusem je dokončení tlačítka pro zobrazení jednobarevného obrázku. Tlačítka jsou zeleně označená v obrázku 2.3.

2.6 Jak odevzdat?

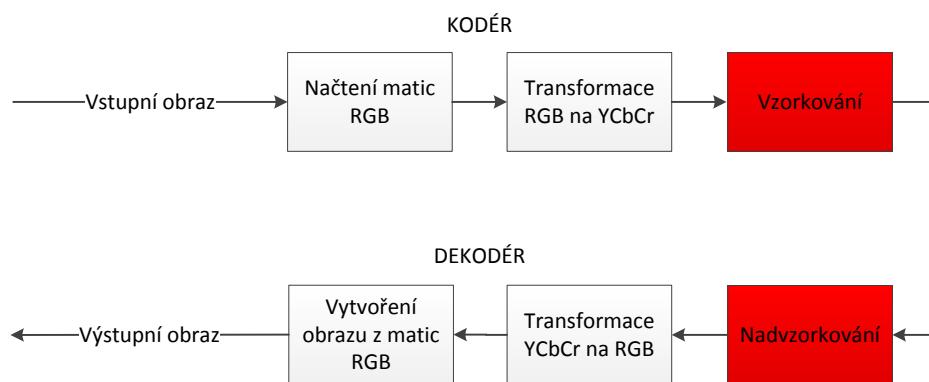
1. Vytvořte adresář, který se bude jmenovat jako Vaše ID.
2. Do adresáře nahrajte obsah vaší složky `src`.
3. Adresář komprimujte do ZIP a vložte ho do e-learningu.

3 Cvičení 3 – Vzorkování

3.1 Cíle cvičení

Cílem druhého cvičení je (viz blokové schéma 3.1):

1. Dokončit implementaci metod pro zobrazení jednotlivých barevných složek modelu RGB a $YCbCr$ z předchozího cvičení.
2. Implementovat metody pro vzorkování modelu $YCbCr$.
3. Implementovat metody pro nadvzorkování modelu $YCbCr$.
4. Propojit metody s rozhraním



Obrázek 3.1: Schéma kodéru i dekodéru

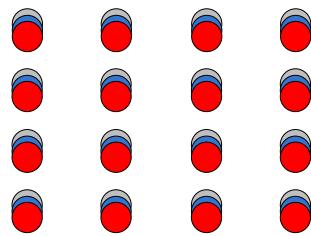
3.2 Teoretická část

Vzorkování modelu $YCbCr$

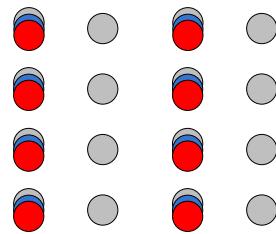
Model lidského zrakového systému HVS (*Human Visual System*) poukazuje na skutečnost, že lidské oko je méně citlivé na barevné složky než na jasové složky obrazu. Z toho důvodu je možné barvy jistým způsobem potlačit a to se děje vzorkováním barvonosných složek. Existují celkem 4 možnosti vzorkování: 4:4:4, 4:2:2 ($YUY2$), 4:2:0 ($YUY12$), 4:1:1.

- **Model 4:4:4** zachovává všechny složky modelu stejné.
- **Model 4:2:2** zachová jasovou složku v původním formátu, podvzorkuje horizontální rozlišení barvonosných složek na polovinu.
- **Model 4:2:0** zachová jasovou složku v původním formátu, podvzorkuje horizontální i vertikální rozlišení barvonosných složek na polovinu.
- **Model 4:1:1** zachová jasovou složku v původním formátu, podvzorkuje horizontální rozlišení barvonosných složek na čtvrtinu.

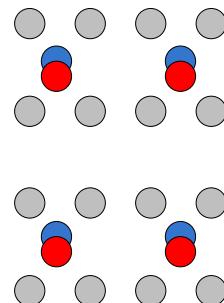
Při vzorkování dochází ke ztrátě informace.



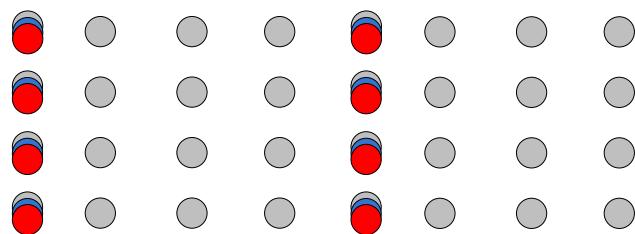
Obrázek 3.2: Model 4:4:4



Obrázek 3.3: Model 4:2:2



Obrázek 3.4: Model 4:2:0



Obrázek 3.5: Model 4:1:1

3.3 Praktická realizace

Dokončete způsob zobrazení jednobarevného obrázku z předchozího cvičení. Následně, podobně jako v předchozím cvičení uděláte novou třídu, která bude obsahovat 2 statické metody pro provedení vzorkování a nadvzorkování. V rozhraní tedy přidáte funkčnost na 2 tlačítka včetně způsobu, jak zvolit typ vzorkování:

- Tlačítko Down Sample pro podvzorkování
- Tlačítko Over Sample pro nadvzorkování
- Způsob pro výběr typu vzorkování (např: radio button, combo box)

3.3.1 Zobrazení komponent barevného modelu RGB

Zobrazení jednotlivých barevných komponent bude od dnešního cvičení dobrou pomůckou. Uvidíme totiž jednotlivé kroky úprav obrázku. Návod, jak barevné složky zobrazit je popsána v předchozím cvičení [2.2.4](#). Nejdůležitějšími složkami pro zobrazení jsou YCbCr složky modifikovaného obrázku, v nich budou úpravy viditelné. Jak vypadá zobrazení jednotlivých barev je v následujících obrázcích: porovnání originálního obrázku a RGB složek (obarvených) na obrázku [3.6](#), dále RGB složky v odstínech sedí [3.7](#) a nakonec YCbCr složky [3.8](#).



Obrázek 3.6: Porovnání barevných složek *RGB* (obarvené)



Obrázek 3.7: Porovnání barevných složek *RGB* (černobílé/šedé)

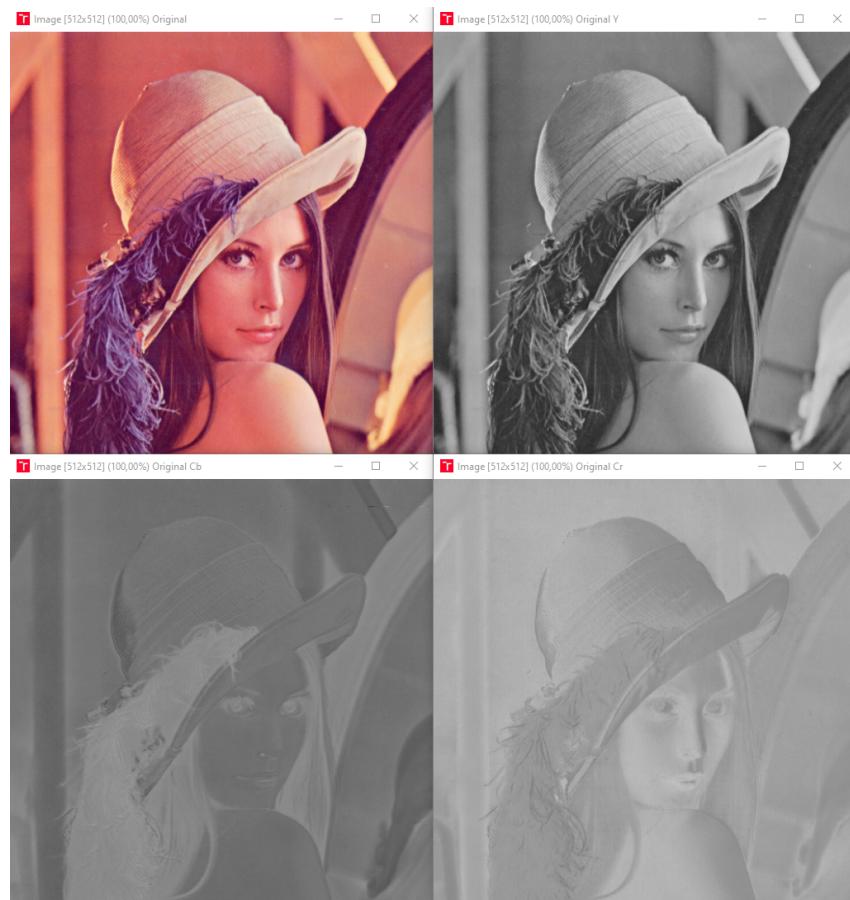
3.3.2 Podvzorkování modelu *YCbCr*

V aplikaci nyní máme 2 tlačítka:

- tlačítko pro podvzorkování,
- tlačítko pro nadvzorkování
- způsob výběru vzorkování

Po stisku tlačítek se provede patřičná akce uvnitř objektu třídy `Process`, kde dojde k zavolání metod, které upraví YCbCr model. Typy vzorkování, které implementujete jsou následující:

- 4:4:4 platí, že všechny matice budou v původní velikosti,
- 4:2:2 platí, že bude vynechán každý druhý sloupec matic *Cb* a *Cr*, barvonosné matice tedy mají výsledné horizontální rozlišení poloviční,
- 4:1:1 platí, že je vždy zachován první sloupec ze čtverice, ostatní jsou vynechány. Barvonosné složky mají výsledné horizontální rozlišení čtvrtinové,
- 4:2:0 platí, že je vynechán každý druhý řádek i sloupec. Barvonosné složky mají poloviční jak horizontální, tak i vertikální rozlišení,



Obrázek 3.8: Porovnání barevných složek $YCbCr$

Samostatná práce

Pro podvzorkování vytvořte v balíčku jpeg novu třídu `Sampling`, uvnitř které budou minimálně tyto 2 veřejné statické metody (obrázek 3.9):

- `sampleDown(Matrix inputMatrix, SamplingType samplingType)`
- `sampleUp(Matrix inputMatrix, SamplingType samplingType)`

```
public static Matrix sampleDown(Matrix inputMatrix, SamplingType samplingType) {
    return sampledMatrix;
}
public static Matrix sampleUp(Matrix inputMatrix, SamplingType samplingType) {
    return sampledMatrix;
}
```

Obrázek 3.9: Metody pro vzorkování (součást testu)

Obě metody budou přijímat a vracet barevnou složku v objektu `Matrix`. Druhý vstupní parametr je Enum třídy `SamplingType`. Tento enum jsme společně vytvořili v prvním cvičení. Pro kontrolu se nachází na obrázku 3.10. Zde je nutné také dodržet minimálně název enumu (hodnoty, které voláme `S_4_4_4`, `S_4_2_2`, `S_4_2_0`, `S_4_1_1`).

```

public enum SamplingType {
    S_4_4_4( s: "4:4:4"), 2 usages
    S_4_2_2( s: "4:2:2"), 2 usages
    S_4_2_0( s: "4:2:0"), 2 usages
    S_4_1_1( s: "4:1:1"); 3 usages

    String name; 2 usages

    SamplingType(String s){ name = s;}

    @Override
    public String toString(){ return "Sampling: " + name;}
}

```

Obrázek 3.10: Enum SamplingType

Pro jednodušší provádění vzorkování je doporučeným postupem udělat si pomocné metody, které provedou část potřebných úprav daných matic. Příkladem může být metoda `private static Matrix downSample (Matrix mat)`, tato metoda bude mít jedinou funkci a to vždy podvzorkovat vstupní matici tak, že z ní vypustí každý druhý sloupec. Výstupní matice tak bude mít poloviční horizontální rozlišení.

Co k tomu můžete použít ze třídy `Matrix`? (Podívejte se do dokumentace v e-learningu, případně stáhněte do IntelliJ)

1. metodu `getRowDimension()`,
2. metodu `getColumnDimension()`,
3. metodu `public void setMatrix(int i0, int i1, int j0, int j1, Matrix X)`.

Uvnitř hlavních metod (které budou testované) `sampleDown` a `sampleUp` bude provedení celé konverze podle zadlého typu vzorkování. Tedy doporučeným postupem je uvnitř obou metod použít `switch/case` a následně volat pomocné metody uvnitř této třídy, které zajistí podvzorkování a nadvzorkování.

Při použití `switch/case` implementujte pro jednotlivé typy patřičné příkazy, které provedou požadované podvzorkování. Po podvzorkování si navrácené matice uložte do patřičných proměnných ve třídě `Process` (`modifiedCb`, `modifiedCr`).

Př: při vzorkování 4:2:0, dochází k vynechání jak řádků, tak i sloupců. K tomu můžete použít metodu `transpose()` ze třídy `Matrix` pro transponování matice. Díky ní můžete opakovaně použít metodu `private Matrix downSample (Matrix mat)`. Pro lepsí pochopení prostudujte obrázek 3.11.

<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	<table border="1"><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr></table>	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	2	3	4	5	6	7	8																																																																																																																																																											
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
<p>Originální matice</p> <p>O_p^T</p>	<p>Podvzorkování</p> <p>O_p^T</p>	<p>Podvzorkovaná matice</p> <p>O_p^T</p>																																																																																																																																																																
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td></tr><tr><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td></tr><tr><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td></tr></table>	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	5	7	7	7	7	7	7	7	7	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td><td>3</td></tr><tr><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td></tr><tr><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td><td>7</td></tr></table>	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	5	7	7	7	7	7	7	7	7	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>3</td><td>3</td><td>3</td><td>3</td></tr><tr><td>5</td><td>5</td><td>5</td><td>5</td></tr><tr><td>7</td><td>7</td><td>7</td><td>7</td></tr></table>	1	1	1	1	3	3	3	3	5	5	5	5	7	7	7	7																																																																																
1	1	1	1	1	1	1	1																																																																																																																																																											
3	3	3	3	3	3	3	3																																																																																																																																																											
5	5	5	5	5	5	5	5																																																																																																																																																											
7	7	7	7	7	7	7	7																																																																																																																																																											
1	1	1	1	1	1	1	1																																																																																																																																																											
3	3	3	3	3	3	3	3																																																																																																																																																											
5	5	5	5	5	5	5	5																																																																																																																																																											
7	7	7	7	7	7	7	7																																																																																																																																																											
1	1	1	1																																																																																																																																																															
3	3	3	3																																																																																																																																																															
5	5	5	5																																																																																																																																																															
7	7	7	7																																																																																																																																																															
<p>Podvzorkovaná matice</p>	<p>Podvzorkování</p>	<p>Podvzorkovaná matice</p>																																																																																																																																																																
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>3</td><td>3</td><td>3</td><td>3</td></tr><tr><td>5</td><td>5</td><td>5</td><td>5</td></tr><tr><td>7</td><td>7</td><td>7</td><td>7</td></tr></table>	1	1	1	1	3	3	3	3	5	5	5	5	7	7	7	7	<table border="1"><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr><tr><td>1</td><td>3</td><td>5</td><td>7</td></tr></table>	1	3	5	7	1	3	5	7	1	3	5	7	<p>Transponovaná O_p^T</p> <p>O_p^T</p> <p>Podvzorkování 4:2:0</p>																																																																																																																																				
1	1	1	1																																																																																																																																																															
3	3	3	3																																																																																																																																																															
5	5	5	5																																																																																																																																																															
7	7	7	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															
1	3	5	7																																																																																																																																																															

Obrázek 3.11: Příklad pro implementaci podvzorkování modelu $YCbCr$ 4:2:0 na konkrétních maticích

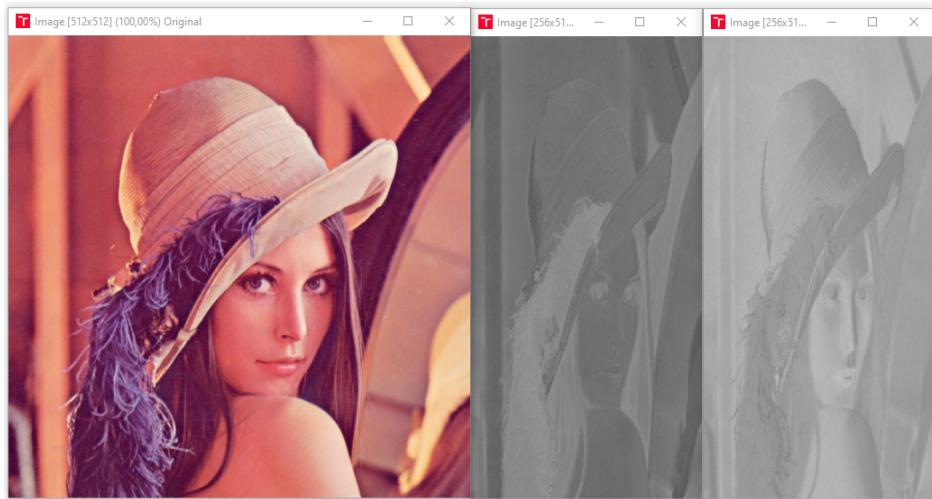
3.3.3 Nadvzorkování modelu YCbCr

Nadvzorkování bude fungovat přesně opačně jako podvzorkování, místo toho, aby se vzal každý druhý sloupec, nyní je potřebné, aby se každý sloupec duplikoval. V nové matici tedy bude první sloupec 2x za sebou, následovat bude druhý sloupec 2x, atd.

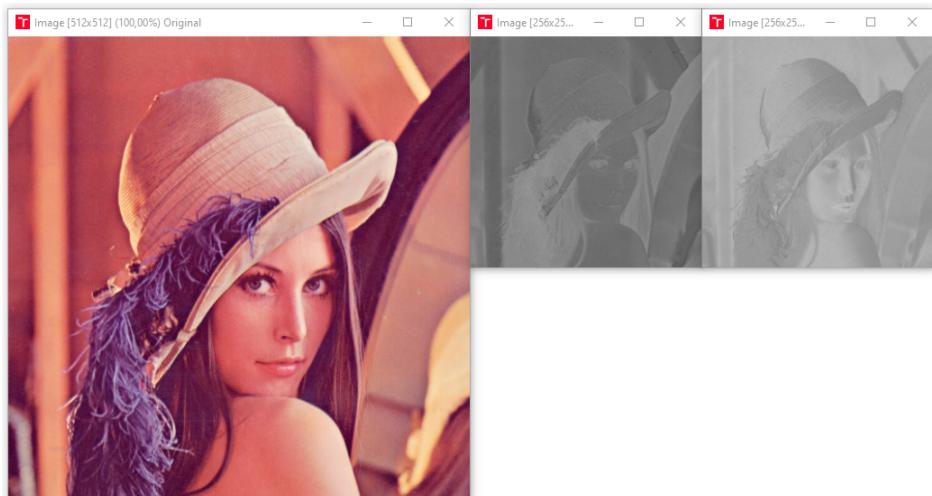
Výsledkem vaší práce by měli být funkční tlačítka pro vzorkování a nadvzorkování, včetně výběru typu vzorkování. Aplikace by také měla být schopná zobrazit vzorkované složky, aby bylo možné vizuálně ověřit, že metody skutečně fungují.

Výsledkem by měla být aplikace schopná zobrazit vzorkované výstupy pro všechny vzorkovací modely viz obrázky 3-12.

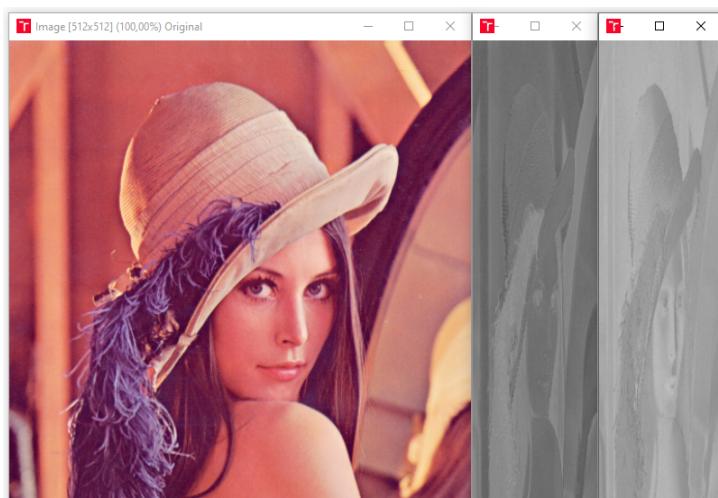
Třídu Sampling otestujete stejným postupem, tak jak je uvedeno v minulém cvičení 2.4, pouhým rozdílem bude v testovacím souboru, který se imenuje SamplingTest.java.



(a) Podvzorkování 4:2:2



(b) Podvzorkování 4:2:0



(c) Podvzorkování 4:1:1

Obrázek 3.12: Vzorové obrázky při podvzorkování modelu $YCbCr$

4 Cvičení 4 – Objektivní hodnocení kvality

4.1 Cíle cvičení

Cílem čtvrtého cvičení je:

1. Implementovat metody pro objektivní hodnocení kvality komprimovaného obrazu.
 - Výpočet MSE
 - Výpočet MAE
 - Výpočet SAE
 - Výpočet PSNR
 - Výpočet SSIM a MSSSIM
2. Úprava rozhraní, pro zobrazení výsledků a volby složky, pro kterou bude kvalita vypočtena.

4.2 Teoretická část

4.2.1 Metody pro měření kvality obrazových dat

Měření a hodnocení vizuální kvality digitálních obrazů a videí je velmi složité, neboť kvalitu ovlivňuje mnoho na sobě nezávislých faktorů. Kvalitu obrazu lze hodnotit buď subjektivně nebo objektivně. Subjektivní měření kvality je velmi závislé na pozorovateli a nelze tedy určit či definovat přesné kvalitativní měřítko. Kvalita obrazu či videa mnohdy závisí na způsobu, jak je s médiem nakládáno (film na DVD, videokonference, identifikační a přístupové systémy apod.). U objektivního měření existují daná kritéria i měřítko, avšak systémy, měřící obrazová data objektivně, nemohou kompletně reprodukovat subjektivní vnímání.

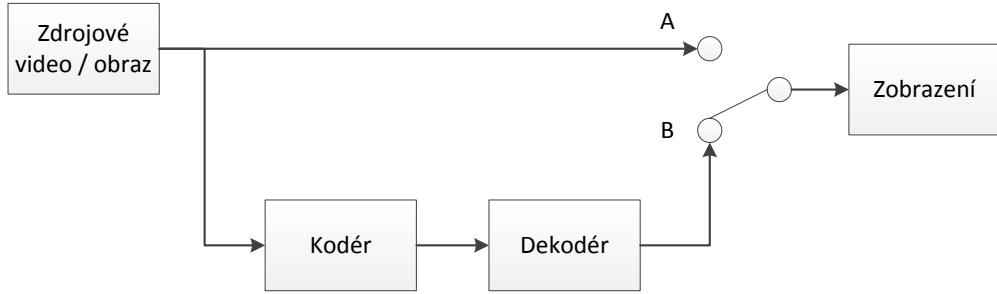
4.2.2 Subjektivní měření kvality

Pro subjektivní měření kvality existuje standard ITU R BT 500-11. Běžně používaná metoda pro měření kvality obrazu ze standardu je metoda DSCQS (*Double Stimulus Continuous Quality Scale*). Při užití této metody je hodnotiteli prezentována dvojice obrazů nebo krátkých videí A a B. Hodnotitel ohodnotí kvalitu na stupnici 1 až 5, kdy 1 je nejhorší a 5 je nejlepší kvalita. Hodnotiteli je standardně zobrazována dvojice, z níž jeden obraz či video je v originální kvalitě a druhý obraz nebo video je modifikováno testovanou metodou viz obrázek 4.1.

Pořadí zobrazení originálního (referenčního) a upraveného snímku/videa je během testování náhodné. Hodnotitel tedy nikdy neví, zda se jedná o originál či upravenou kopii. Tím je zamezeno ovlivňování hodnotitele. Na konci testování jsou výsledky konvertovány do normalizovaného rozsahu. Výsledek, často označován jako MOS (*Mean Opinion Score*), indikuje relativní kvalitu referenčního a upraveného videa/obrazu.

Subjektivní test se v praxi potýká s následujícími problémy:

1. Výsledek velmi záleží na subjektivních pocitech každého hodnotitele.
2. Výsledek velmi záleží na testované videosekvenci/obrazu.
3. Expert v oblasti kompresí videosekvencí/obrazu může hodnotit neobjektivně.
4. Testy jsou finančně nákladné.



Obrázek 4.1: Testovací systém dle DSCQS

4.2.3 Objektivní měření kvality

Objektivní měření kvality lze na rozdíl od subjektivního měření kvality plně automatizovat algoritmizací. Tím je měření podstatně levnější. Vývojáři kompresních technik obrazu a videa často používají objektivní techniky díky rychlosti a jednoduchosti implementace a možnosti reálného srovnání výsledků s konkurencí. Jedním z nejrozšířenějších algoritmů pro objektivní hodnocení kvality obrazových dat je metoda PSNR (*Peak Signal to Noise Ratio*). Dále se používá MSE (*Mean Squared Error*), MAE (*Mean Absolute Error*), popřípadě SAE (*Sum of Absolute Errors*)

Peak Signal to Noise Ratio – PSNR

PSNR se měří na logaritmické stupnici a závisí na střední kvadratické chybě MSE (*Mean Squared Error*) mezi originálním a upraveným obrazem nebo snímkem videa vztažené k hodnotě $(2^n - 1)^2$, kde n je počet bitů na jeden obrazový bod. PSNR se vypočítá pomocí vztahu

$$PSNR = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} [dB], \quad (4.1)$$

kde MSE je dáno sumou

$$MSE = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} [x(m, n) - x'(m, n)]^2, \quad (4.2)$$

kde

- M je šířka obrazu,
- N je výška obrazu,
- x je obrazový bod originálu,
- x' je obrazový bod upraveného obrazu.

Pro PSNR platí následující:

- Pro výpočet PSNR je potřeba originální obraz, který nemusí být vždy k dispozici.
- Hodnota PSNR nezcela koreluje se subjektivní kvalitou podle ITU-R 500.

- Vysoká hodnota PSNR obvykle indikuje vysokou kvalitu obrazu, nízká hodnota PSNR obvykle indikuje nízkou kvalitu obrazu.

Oba výpočty (PSNR a MSE) jsou určené pro výpočet monochromatického obrazu (jednobarevného). Pro použití na barevný obraz je nutné MSE vypočítat pro každou barvu zvlášť a pro tyto výsledky spočítat aritmetický průměr (sečist a podělit třemi). Ve výpočtu PSNR zůstane hodnota n rovna 8 a MSE je výsledný aritmetický průměr z MSE hodnot barevných složek.

Mean Absolute Error – MAE

MAE se vypočítá jako průměr absolutních hodnot rozdílů mezi originálním a upraveným (komprimovaným) obrazem. Čím menší MAE je, tím je kvalita upraveného obrazu vyšší. Výpočet se provede dle rovnice

$$MAE = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |x(m, n) - x'(m, n)|, \quad (4.3)$$

kde M je šířka, N je výška obrazu, X je matice jedné barevné složky originálního obrazu a X' je matice korespondující barevné složky upraveného obrazu.

Sum of Absolute Errors – SAE

SAE se vypočítá jako součet absolutních hodnot rozdílů mezi originálním a upraveným (komprimovaným) obrazem. Čím menší SAE je, tím je kvalita upraveného obrazu vyšší. Výpočet se provede dle rovnice

$$SAE = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |x(m, n) - x'(m, n)| \quad (4.4)$$

kde X je matice jedné barevné složky originálního obrazu a X' je matice korespondující barevné složky upraveného obrazu.

SAE se velmi často používá při porovnání dvou obrazů při detekci pohybu ve videu.

Structural Similarity Index – SSIM

SSIM narozdíl od ostatních objektivních metod pracuje se strukturálním rozpoložením měřených dat. Vyjádřená podobnost by tedy měla odpovídat tomu, jak rozdíl vnímá lidské oko. Metoda pracuje s předpokladem, že jas na povrchu objektu je produktem osvětlení a odrazu, ale struktura objektů je na jasu nezávislá. Snahou tvůrců této metody je odstranit vlivy osvětlení na výsledky měření podobnosti. SSIM index dosahuje hodnot od -1 do 1, je počítán jen pro jasové složky a je vyjádřen rovnicí

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (4.5)$$

kde x a y představují originální a upravená data, μ průměr jasových hodnot, σ střední kvadratickou odchylku, σ_{xy} kovarianci a konstanty C_1 , C_2 stabilizují výpočet, pokud se

jmenovatel příliš blíží k nule [?].

Pro výpočet μ_x , σ_x a σ_{xy} jsou využity rovnice:

$$\mu_x = \frac{1}{N} \sum_{i=1}^N x_i, \quad (4.6)$$

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2 \right)^{\frac{1}{2}}, \quad (4.7)$$

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y), \quad (4.8)$$

kde N představuje počet pixelů, x_i , y_i jsou hodnoty pixelů a μ_x , μ_y vypočítané průměrné hodnoty jasových koeficientů.

Hodnoty konstant C_1 a C_2 jsou určeny za pomocí rovnic:

$$C_1 = (K_1 L)^2, \quad (4.9)$$

$$C_2 = (K_2 L)^2, \quad (4.10)$$

kde L představuje dynamický rozsah hodnot jednotlivých pixelů (při 8-bitovém zobrazení je rovna 255). Konstanty K_1 a K_2 musí odpovídat hodnotám mnohem menším než 1. Jsou proto voleny následovně: $K_1 = 0,01$, $K_2 = 0,03$ [?].

Index SSIM je možné vypočítat pro porovnání dvou celistvých jasových složek. Autoři tento postup ale nedoporučují, navrhují oba obrazové signály rozdělit na pole o velikosti 8x8 px a SSIM index spočítat pro každé dvě odpovídající si pole zvlášt. Aritmetickým průměrem těchto údajů je možné vypočítat střední hodnotu MSSIM (Mean SSIM) dle rovnice:

$$MSSIM(x, y) = \frac{1}{M} \sum_{j=1}^M SSIM(x_j, y_j), \quad (4.11)$$

kde M představuje počet vypočtených dílčích indexů SSIM a $SSIM(x_j, y_j)$ jsou hodnoty těchto indexů.

4.3 Praktická realizace

Cílem samostatné práce jsou následující úkoly:

- Vytvořit třídu Quality, která bude provádět potřebné výpočty
- Třída Quality bude schopna vypočítat MSE, MAE, SAE, PSNR a volitelně SSIM a MSSIM
- Rozšířit rozhraní pro výběr a výpis vypočtených hodnot

4.3.1 Implementace algoritmů kvality

Vytvořte si novou třídu s názvem Quality v balíčku jpeg, ve které budete implementovat vzorce pro MSE, MAE, SAE a PSNR. Všechny metody zobrazené na obrázku 4.2 máte dostupné v e-learningu. Obsah metod pro MSE, MAE a SAE je v podstatě shodný, liší se pouze prostřední část (rozdíl mezi MSE, MAE a SAE). Za zmínu také stojí to, že tyto metody používají vstupní pole typu double [] []. Při použití složek z YCbCr použijete pro převod z Matrix na double [] [] metodu matrix.getArray(). Problém může nastat při použití polí z RGB složek (ty máme typu int [] []). Pro jednoduchou konverzi můžete použít následující kód:

```
public static double[][] convertInttoDouble(int[][] intArray) {

    double[][] doubleArray = new double[intArray.length][intArray[0].length];
    for (int i = 0; i < intArray.length; i++) {
        for (int j = 0; j < intArray[0].length; j++) {
            doubleArray[i][j] = (double) intArray[i][j];
        }
    }

    return doubleArray;
}
```

Ve třídě Quality si implementujte i metody pro SSIM a MSSIM. Jejich výpočet a samotnou implementaci si ponechte až nakonec. Do té doby, ponechте výstup metod formou throw new RuntimeExceptoin("Not implemented yet."). Unit test s tímto počítá a nebude tyto metody kontrolovat, dokud budou vracet tento exception.

Právě pro výpočet kvality jsme si v prvních hodinách ve třídě Process vytvořili proměnné, které nám drží zachované původní hodnoty všech složek: R, G, B a Y, Cb, Cr. Nyní je možné tyto hodnoty porovnat s modifikovanými hodnotami.

4.3.2 Úprava grafického rozhraní

Obrázek 4.3 představuje návrh změn v rozhraní pro zobrazení všech potřebných hodnot pro výsledky výpočtů z objektivního hodnocení kvality. Zelenou část implementujete všichni, oranžovou část představuje bonusovou část samostatné práce. Potřebné výpočty se provedou po stisknutí tlačítka Count PSNR a Count SSIM (zavolání potřebných metod ve třídě Process a předání potřebných parametrů). Výpočty se provedou pouze na zvolené barevné složce (aby bylo možné porovnat, kde se změny projevují nejvíce).

```

//Výpočet MSE. Nutný převod pomocí Matrix.getArray() nebo převod z int[][] na double[][].
public static double countMSE(double [][] original, double [][] modified) { 1 usage
    return mse;
}

//Výpočet MAE. Nutný převod pomocí Matrix.getArray() nebo převod z int[][] na double[][].
public static double countMAE(double [][] original, double [][] modified) { 1 usage
    return mae;
}

//Výpočet SAE. Nutný převod pomocí Matrix.getArray() nebo převod z int[][] na double[][].
public static double countSAE(double [][] original, double [][] modified) { 1 usage
    return sae;
}

//Výpočet PSNR z MSE
public static double countPSNR(double MSE) { 1 usage
    return psnr;
}

//Výpočet PSNR z MSE pro RGB obrázek (průměr z barev a použít např. countPSNR)
public static double countPSNRforRGB(double mseRed, double mseGreen, double mseBlue) { 1 usage
    return psnrRGB;
}

//Výpočet SSIM, ponechte zde throw, pokud to nebudete implementovat
public static double countSSIM(Matrix original, Matrix modified) { 1 usage
    throw new RuntimeException("Not implemented yet.");
}

//Výpočet MSSIM, ponechte zde throw, pokud to nebudete implementovat
public static double countMSSIM(Matrix original, Matrix modified) { 1 usage
    throw new RuntimeException("Not implemented yet.");
}

```

Obrázek 4.2: Metody pro Unit test

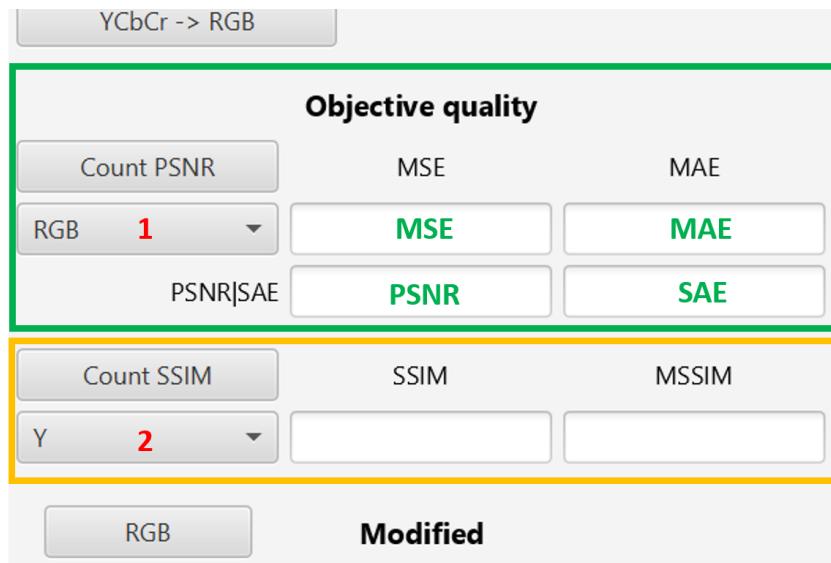
K volbě barevné složky je doporučeným ovládacím prvkem **ComboBox**, který naplníte nově vytvořenou **enum** třídou (například **QualityType**). Volby pro výpočet PSNR (box {1}) bude obsahovat následující možnosti:

- Jednotlivé složky: Red, Green, Blue
- Jednotlivé složky: Y, Cb, Cr
- RGB – výpočet na všech složkách současně
- YCbCr – výpočet na všech složkách současně

Pokud bude zvolený výstup RGB nebo YCbCr, tak v polích pro MSE, MAE a SAE zobrazte průměrnou hodnotu vypočtenou ze všech 3 složek.

Ovládací prvek pro výpočet SSIM (box {2}) bude obsahovat pouze hodnoty Y, Cb a Cr (můžete použít stejný **enum**, pouze vložit jen některé hodnoty).

Pozor: výpočet kvality je možné provést až ve chvíli, kdy máte provedené (naplněné) potřebné hodnoty. Například výpočet na RGB bude pravděpodobně možné provést až po zpětném převodu z YCbCr. Důležité je tedy vědět, kdy mohu provést výpočet kvality na kterých složkách.



Obrázek 4.3: Návrh ovládání v GUI pro kvalitu

Vložení hodnoty do textového pole

Do textového pole v grafickém rozhraní (pole musí mít nastavené fx:id) lze vložit text několika způsoby. Největším problémem v našem případě je to, že **Controller** rozhraní nemá přímý přístup k vypočteným hodnotám. Nastavit text je tedy možné pomocí těchto dvou základních způsobů:

1. `textPole.setText("nova hodnota");`
2. Pomocí proměnné, která je přímo provázána s textovým polem, jakákoli taková hodnota je typu **Property**.
 - Z pole získáme tuto proměnnou: `textPole.textProperty();`
 - Můžeme si ji předat do metody z třídy **Process**
 - Následně nastavit hodnotu této proměnné `textProperty.set("nova hodnota");`
 - Hodnota se tím tak ihned v GUI změní¹

Pozn: Dále by bylo vhodné například smazat obsah textového pole ve chvíli, kdy změníme složku, pro kterou chceme kvalitu vypočítat, případně výpočet zavolat automaticky po výběru nové složky pro zobrazení.

Kontrola kódu

Zkontrolujte si správnou funkčnost třídy **Quality** pomocí Unit testu, který máte dostupný v e-learningu (návod viz kapitola 2.4)

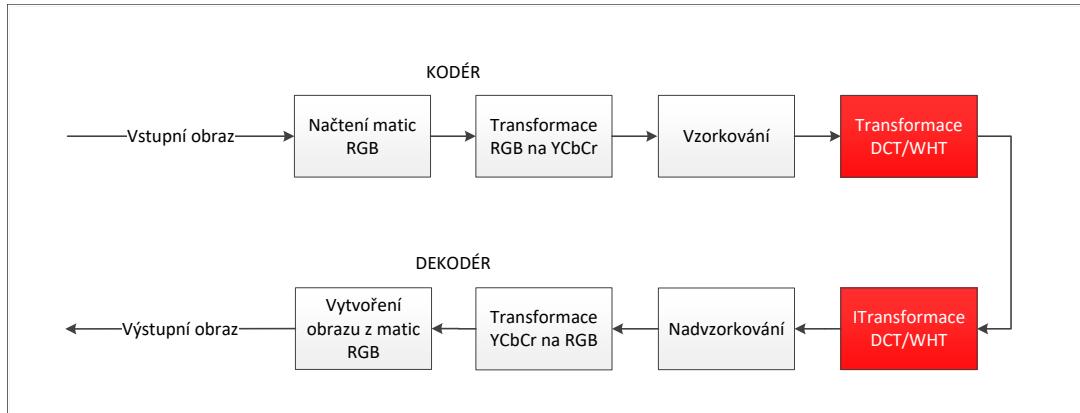
¹Může zde nastat problém, pokud se tato změna provede na jiném vlákně, než ve vlákně, které je určené pro JavaFX. V našem případě toto nenastane, protože všechny výpočty provádí vláknko, které se zároveň stará o vykreslování rozhraní. Ve většině případů je ale nutné oddělit vlákna rozhraní od pracovních vláken. Pokud by vláknu trval výpočet moc dlouho, tak by rozhraní zamrzlo a OS by u okna poznamenal, že aplikace neodpovídá.

5 Cvičení 5 – Transformace

5.1 Cíle cvičení

Cílem čtvrtého cvičení je (viz blokové schéma 5.1):

1. Implementovat metody pro tvorbu transformačních matic (DCT a WHT).
2. Implementovat obecné metody pro 2-D transformaci (DCT a WHT).
3. Zpracovat celý obraz s proměnlivou velikostí bloků



Obrázek 5.1: Schéma kodéru i dekodéru

5.2 Teoretická část

5.2.1 Transformační kódování

Transformační kódování je důležitou součástí video kodeků. Vzorky z prostorové oblasti jsou transformovány do jiné reprezentace (transformační domény). Transformace se používá hlavně z toho důvodu, že sousední vzorky v prostorové oblasti jsou vysoce korelovány a celková energie je rozložena do celého snímku. Díky tomu je velmi těžké data v prostorové oblasti redukovat bez ztráty kvality výsledného obrazu. Při vhodně zvolené transformaci se data "lépe" komprimují bez znatelné ztrátě na obrazové kvalitě. Transformační kódování koncentruje energii do malého počtu vzorků, které jsou poté velmi důležité – dekoreluje data.

Velmi rozšířené transformace používané v obrazových datech jsou:

1. Diskrétní kosinová transformace (DCT) – obvykle je aplikovaná na menší bloky (například velikosti 8x8 pixelů). Používá se v řadě současných standardů – JPEG, H.261, H.263, H.263+, MPEG-2, MPEG-4.
2. Diskrétní vlnková (waveletova) transformace (DWT) – obvykle aplikovaná na větší plochy obrazu nebo na celý obraz. Používá se v nových standardech – JPEG-2000, MPEG-4 pro statické obrazy.

Transformace, které jsou využívány při kompresi obrazových dat jsou vždy lineární, což znamená, že pro získání sekvence transformovaných koeficientů θ ze sekvence vstupních hodnot x_n použijeme vztah

$$\theta_n = \sum_{N-1}^{i=0} x_i a_{n,i}, \quad (5.1)$$

který definuje vztah pro výpočet dopředné transformace. Původní sekvenci x_n získáme z transformované sekvence θ inverzní transformací definovanou vztahem

$$x_n = \sum_{N-1}^{i=0} \theta_i b_{n,i}, \quad (5.2)$$

Transformace mohou být přepsány do maticové formy následovně:

$$\theta = Ax, \quad (5.3)$$

$$x = B\theta, \quad (5.4)$$

kde A a B jsou matice velikosti $N \times N$. Jednotlivé elementy matice jsou dány jako

$$[A]_{i,j} = a_{i,j}, \quad (5.5)$$

$$[B]_{i,j} = b_{i,j}. \quad (5.6)$$

Matice A a B jsou transformační matice pro dopřednou a zpětnou transformaci a jsou navzájem inverzní, což znamená, že $AB = BA = I$, kde I je jednotková matice.

Předchozí vztahy se věnují jednorozměrné transformaci, která našla své uplatnění například při zpracování hlasu či audia. Pro digitální obrazy a video se využívá transformace dvou-dimenzionální.

Obecná transformace pro 2D blok obrazu o velikosti $N \times N$ je dána vztahem

$$\Theta_{k,l} = \sum_{N-1}^{i=0} \sum_{N-1}^{j=0} X_{i,j} a_{i,j,k,l}. \quad (5.7)$$

Veškeré 2D transformace využívané v dnešní době pracují způsobem, kdy se transformovaný blok nejprve transformuje pomocí 1D transformace v jednom směru a poté opakujeme 1 D transformaci v druhém směru. U maticového výpočtu je tedy 1D transformace aplikována nejprve na sloupce a výsledek je dále transformován po řádcích. Transformace je reprezentována vztahem

$$\Theta_{k,l} = \sum_{N-1}^{i=0} \sum_{N-1}^{j=0} a_{k,i} X_{i,j} a_{l,j}. \quad (5.8)$$

Pomocí matic lze dopředná transformace vztah vyjádřit vztahem

$$\Theta = AXA^T, \quad (5.9)$$

zpětná pak vztahem

$$X = A^T \Theta A. \quad (5.10)$$

Transformace, kterými se zabýváme a budeme zabývat budou ortonormální. Ortonormální transformace má tu vlastnost, že inverzní transformační matice je vyjádřená transponovanou transformační maticí.

Diskrétní kosinová transformace

Diskrétní kosinova transformace se používá v obrazových a video kodecích díky tomu, že dokáže efektivně transformovat obraz do podoby, která je vhodná ke kompresi a zároveň proto, že je vhodná jak k softwarové, tak i hardwarové implementaci. Dopředná transformace DCT (FDCT) transformuje množinu prostorových vzorků na množinu transformačních koeficientů, zpětná (inverzní) transformace DCT (IDCT) transformuje množinu transformačních koeficientů na množinu obrazových vzorků. Transformace jsou při kompresi obrazu a videa běžně užívány v podobě 1-D a 2-D.

Transformace DCT má při použití na obrazová data 2 dobré vlastnosti:

- zhuštění energie do malého počtu koeficientů,
- dekorelace vstupních dat.

Transformační matice libovolné velikosti $N \times N$ lze odvodit z následujících vztahů:

- pro $i = 0$ a $j = 0, 1, \dots, N - 1$ platí

$$|C|_{i,j} = \sqrt{\frac{1}{N}} \cos \frac{(2j+1)i\pi}{2N}, \quad (5.11)$$

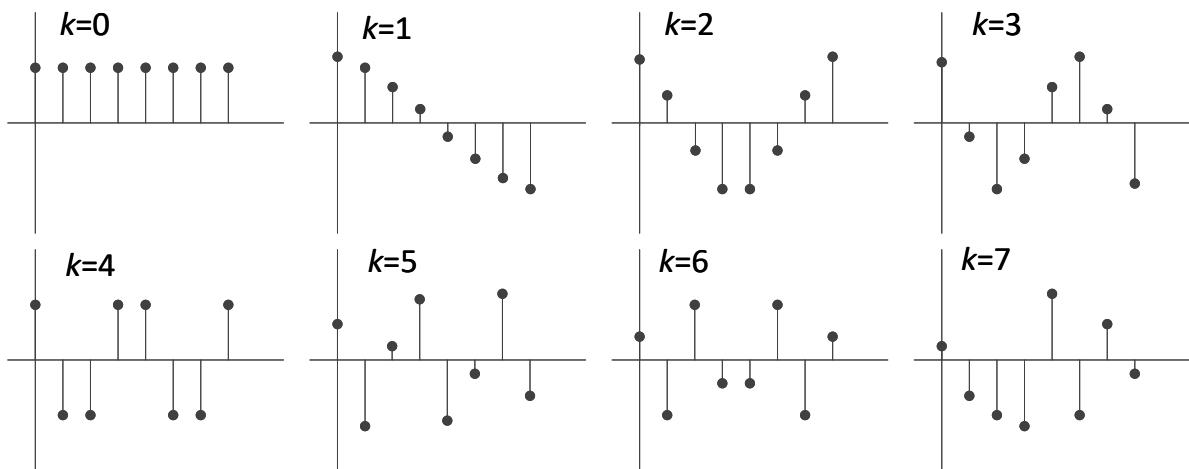
- pro $i = 1, 2, \dots, N - 1$ a $j = 0, 1, \dots, N - 1$ platí

$$|C|_{i,j} = \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N}. \quad (5.12)$$

Pro příklad je uvedena v tabulce 5.1 transformační kosinová matice o rozměru 8×8 (graficky na obrázku 5.2). [?]

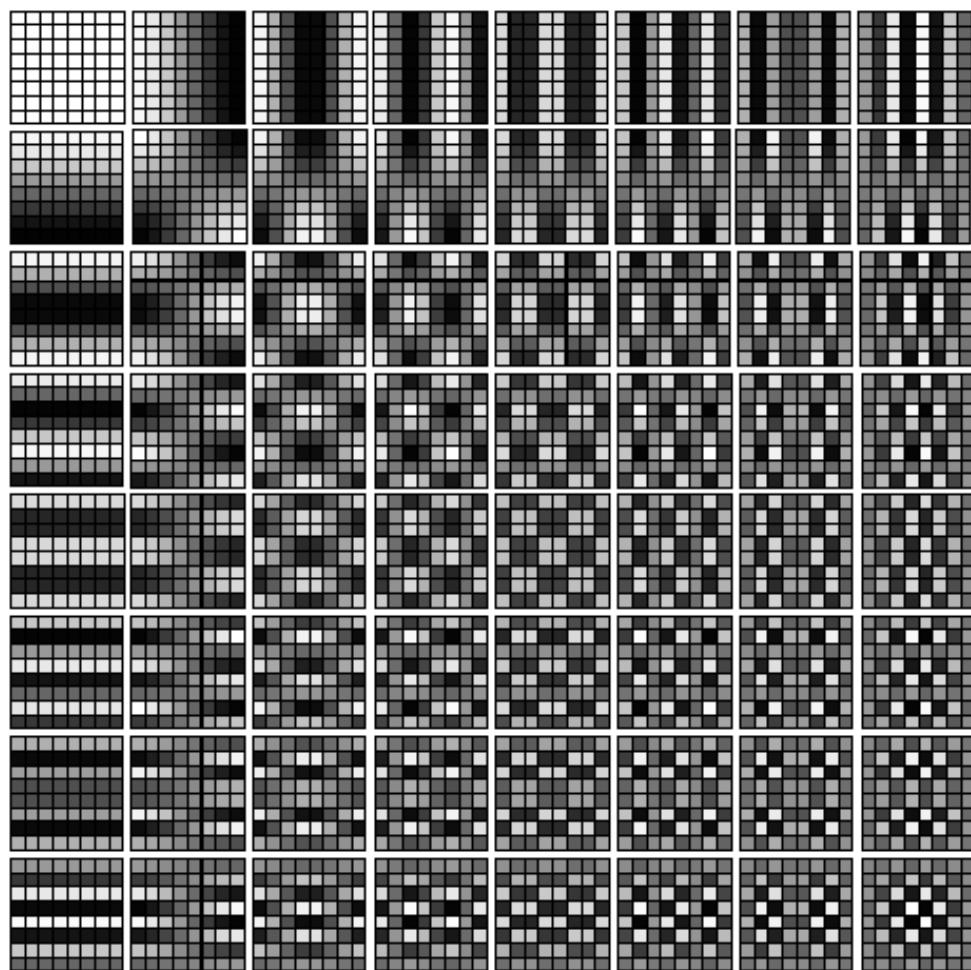
$ C _{i,j}$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
$i = 0$	0,35	0,35	0,35	0,35	0,35	0,35	0,35	0,35
$i = 1$	0,49	0,42	0,28	0,10	-0,10	-0,28	-0,42	-0,49
$i = 2$	0,46	0,19	-0,19	-0,46	-0,46	-0,19	0,19	0,46
$i = 3$	0,42	-0,10	-0,49	-0,28	0,28	0,49	0,10	-0,42
$i = 4$	0,35	-0,35	-0,35	0,35	0,35	-0,35	-0,35	0,35
$i = 5$	0,28	-0,49	0,10	0,42	-0,42	-0,10	0,49	-0,28
$i = 6$	0,19	-0,46	0,46	-0,19	-0,19	0,46	-0,46	0,19
$i = 7$	0,10	-0,28	0,42	-0,49	0,49	-0,42	0,28	-0,10

Tabulka 5.1: Transformační kosinová matice 8x8



Obrázek 5.2: Základní množina signálů pro diskrétní kosinovou transformaci.

Jak je patrné z obrázku 5.2, frekvence opakování kosinové vlnky narůstá s každým řádkem. Jiný pohled na vlastnosti kosinové transformace může přiblížit obrázek 5.3.



Obrázek 5.3: Základní matice pro 2D-DCT o velikosti 8×8

Diskrétní kosinová transformace je velmi významnou při komprezi digitálního obrazu a videa a je využívána mimo jiné ve standardech JPEG, MPEG, H.26x.

Diskrétní Walsh-Hadamardova transformace

Implementace Walsh-Hadamardovy transformace je velmi jednoduchá. Transformační matice je vytvořena z Hadamardovy matice H , která je čtvercová, má velikost $N \times N$, když N je vždy mocninou čísla 2. Hadamardova matice může být sestavena následujícím způsobem:

$$H_N = \begin{bmatrix} H_{(N/2)} & H_{(N/2)} \\ H_{(N/2)} & -H_{(N/2)} \end{bmatrix}, \quad (5.13)$$

když $[H_1] = [1]$. Matice větších rozměrů mají tedy následující tvar:

$$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad (5.14)$$

$$H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad (5.15)$$

$$H_8 = \begin{bmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}. \quad (5.16)$$

Transformační matice pro diskrétní Walsh-Hadamardovu transformaci se získá normalizací Hadamardových matic koeficientem $\frac{1}{\sqrt{N}}$. Matice po normalizaci mají tedy tvar:

$$H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad (5.17)$$

$$H_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad (5.18)$$

$$H_8 = \frac{1}{\sqrt{8}} \begin{bmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}. \quad (5.19)$$

Diskrétní Walsh-Hadamardova transformace se díky své jednoduchosti, dané hodnotami +/- 1 (bez normalizačního faktoru), využívá zejména tam, kde je potřeba minimalizovat výpočetní náročnost.[?]

5.3 Praktická realizace

Praktickou část je možné rozdělit do několika kroků:

1. Vytvoření nové třídy pro všechny metody.
2. Vytvoření metody pro získání transformačních matic.
3. Implementace metody pro tvorbu DCT transformační matice libovolných rozměrů.
4. Implementace metody pro tvorbu WHT transformační matice libovolných rozměrů.
5. Vytvoření metody pro transformaci.
6. Vytvoření metody pro zpětnou transformaci.
7. Propojení metod se třídou `Process` a s grafickým rozhraním.
8. Úprava transformací, aby bylo možné obrázek zpracovat po blocích.

5.3.1 Vytvoření třídy a kostry metod

V balíčku `jpeg` vytvořte třídu `Transform`, která bude obsahovat minimálně tyto 3 statické metody (viz obrázek 5.4). Všechny metody používají enum `TransformType`, který jsme vytvořili v první hodině (měl by obsahovat položky: `DCT("DCT")` a `WHT("WHT")`).

```
//Transformace předané barevné složky
public static Matrix transform(Matrix input, TransformType type, int blockSize) {
    return output;
}

//Inverzní transformace
public static Matrix inverseTransform(Matrix input, TransformType type, int blockSize) {
    return output;
}

//Získání transformační matice, generované podle typu a velikosti bloku
public static Matrix getTransformMatrix(TransformType type, int blockSize) {
    return transformMatrix;
}
```

Obrázek 5.4: Metody pro Unit test

5.3.2 Implementace metod pro vytvoření transformačních matic libovolných rozměrů

Pro generování transformačních matic si vytvořte pomocné metody ve třídě `Transform`. Výslednou matici získáme metodou `getTransformMatrix(TransformType type, int blockSize)`. Tato metoda podle typu rozhodne, která transformační matice se vygeneruje (DCT vs WHT). Výstupem metody bude `Matrix` s transformační maticí zadанého rozměru (velikost bude ve formátu 2^n , kde n je v rozmezí 1–9, ale je možné použít i větší rozměr).

DCT matici vygenerujete pro zadanou velikost bloku pomocí vzorců 5.11 a 5.12. U vzorců nezapomeňte na skutečnost, že v první rovnici pro první řádek se liší část výpočtu. (Pro matematické funkce používejte třídu `Math`).

WHT matici vygenerujte také podle zadané velikosti bloků (stejné hodnoty jako u DCT). Použijte postup, který je popsán v teoretické části. **POZOR – nezapomeňte matici násobit koeficientem $1/\sqrt{N}$.** Pro násobení matice skalární hodnotou můžete použít metodu `times(double s)` ze třídy `Matrix`.

5.3.3 Implementace obecné metody pro 2D transformaci

Transformace naprogramujete pomocí vzorců pro 2D transformaci pomocí maticového roznásobení. K násobení použijte metodu `times()`. Dávejte pozor na dodržení pořadí násobených matic! Metody budou funkční pro oba typy transformací. Jediný rozdíl je v použité transformační matici.

Pro transformaci barevné složky implementujte ve třídě `Transform` metodu `public static Matrix transform(Matrix input, TransformType type, int blockSize)`. Tato metoda si získá potřebnou transformační matici a aplikuje potřebné zpracování pro transformaci obrázku (viz rovnice 5.9).

Zpětnou transformaci provedete podobným postupem v metodě `public static Matrix inverseTransform(Matrix input, TransformType type, int blockSize)`. Zde aplikujete rovnici 5.10.

5.3.4 Propojení se zbytkem aplikace

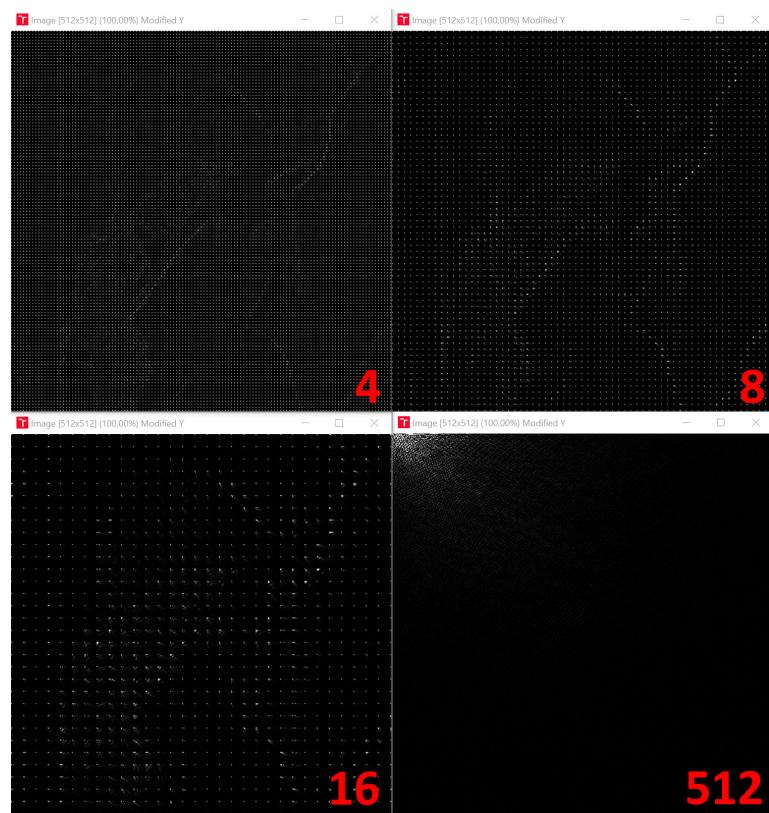
Ve třídě `Process` vytvořte metodu, kterou vyvoláte stiskem tlačítka v GUI, která provede DCT/WHT transformaci na celém obraze. Transformace bude probíhat na složkách Y, Cb, Cr a do těchto složek se také uloží.

5.3.5 Provedení transformace rozdělením obrazu na bloky velikosti $N \times N$

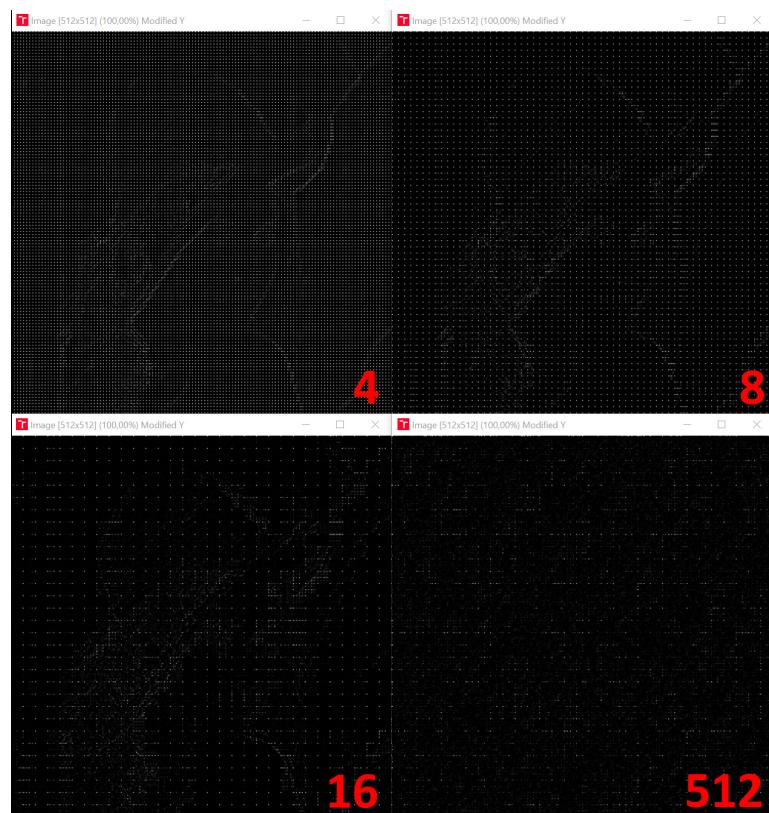
Nyní upravte metody pro transformaci a pro zpětnou transformaci tak, aby bylo možné provést transformaci obrázku v jednotlivých blocích o velikosti $N \times N$, kde N je mocninou čísla 2. Každý takový blok bude následně transformován zvlášť. Uvnitř obou metod stačí vytvořit shodný kód, který pomocí dvou for cyklů (procházení po řádcích a sloupcích se skokem o velikost bloku) je možné celou matici vyčíst po blocích. K provedení je možné použít metody objektu `Matrix`:

- `getMatrix(row, finalRow, col, finalCol)` – tím získáte submatici, kde zadáváte levý horní a pravý dolní index/kraj submatice.
- `setMatrix(row, finalRow, col, finalCol, matrixToInsert)` – tím lze vložit submatici do vybrané části jiné matice.

Jak bude vypadat výstup například Y složky se můžete podívat na obrázek 5.5 pro DCT transformaci a na obrázek 5.6 pro WHT transformaci.



Obrázek 5.5: Ukázka DCT pro bloky o velikosti 4, 8, 16 a 512



Obrázek 5.6: Ukázka WHT pro bloky o velikosti 4, 8, 16 a 512