



# 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

<b>0 Úvod</b>	<b>2</b>
0.1 Instalace vývojového prostředí knihoven . . . . .	2
0.2 Základy práce s vývojovým prostředím IntelliJ IDEA . . . . .	2
0.3 Instalace SceneBuilder . . . . .	2
<b>1 Cvičení 1 – Úvod do JavaFX</b>	<b>3</b>
1.1 Vytvoření projektu . . . . .	3
1.2 Tvorba rozhraní . . . . .	4
1.3 Spuštění hlavního okna . . . . .	5
<b>2 Cvičení 2 – Převody barevných modelů</b>	<b>6</b>
2.1 Teoretická část . . . . .	7
2.1.1 Barevný model <i>RGB</i> . . . . .	7
2.1.2 Barevný model <i>YCbCr</i> . . . . .	8
2.2 Praktická část . . . . .	9
2.2.1 Načtení a zobrazení digitálního obrazu . . . . .	9
2.2.2 Metoda pro získání matic <i>RGB</i> z načteného obrazu . . . . .	10
2.2.3 Metoda pro vytvoření nového obrazu z matic <i>RGB</i> . . . . .	11
2.2.4 Zobrazení pouze jedné barevné složky . . . . .	11
2.2.5 Ověření funkčnosti implementovaných metod . . . . .	12
2.3 Samostatná práce . . . . .	12
2.3.1 Na co si dát pozor? . . . . .	13
2.4 Jak otestovat? . . . . .	13
2.5 Bonus . . . . .	14
2.6 Jak odevzdat? . . . . .	14
<b>3 Cvičení 3 – Vzorkování</b>	<b>15</b>
3.1 Cíle cvičení . . . . .	15
3.2 Teoretická část . . . . .	15
3.3 Praktická realizace . . . . .	17
3.3.1 Zobrazení komponent barevného modelu <i>RGB</i> . . . . .	17
3.3.2 Podvzorkování modelu <i>YCbCr</i> . . . . .	18
3.3.3 Nadvzorkování modelu <i>YCbCr</i> . . . . .	21

# 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

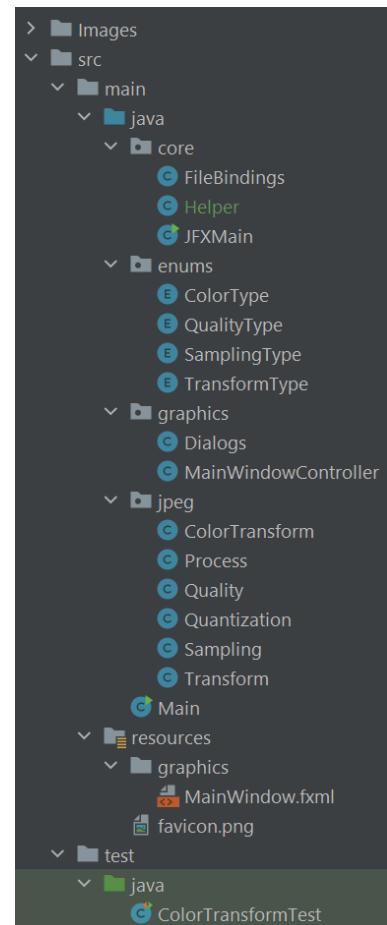
V IntelliJ IDEA si založte nový projekt. Doporučený postup: **File -> New -> Project**. V novém okně stačí vybrat nejzákladnější možnost, tedy **New Project** a zde si nastavit název, umístění, verzi Javy (**Java 11** je v učebně). Poslední je **Build system** ten nastavte na **Maven**.

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>
```

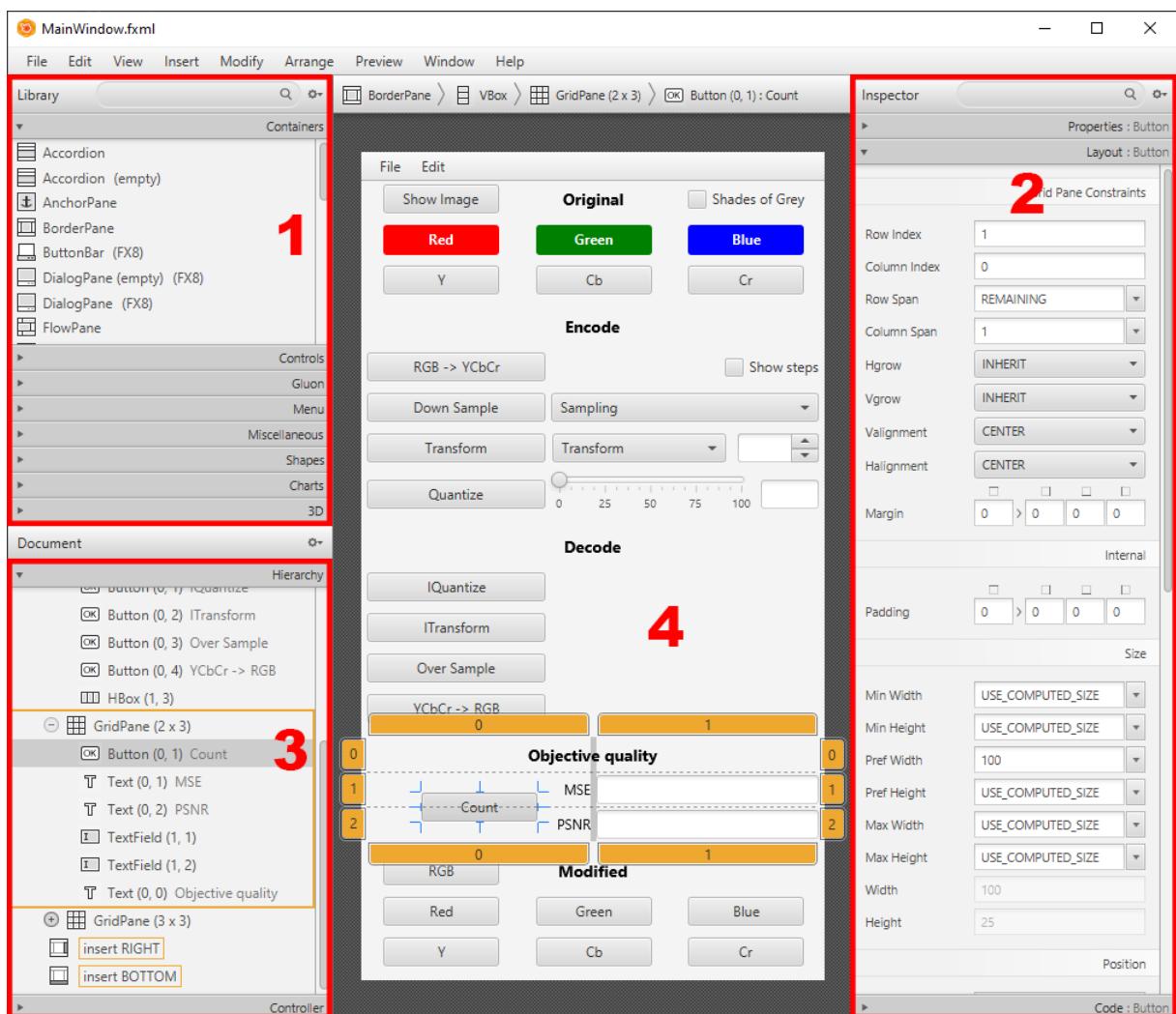


Obrázek 1.1: Struktura

## 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);
    }
}
```

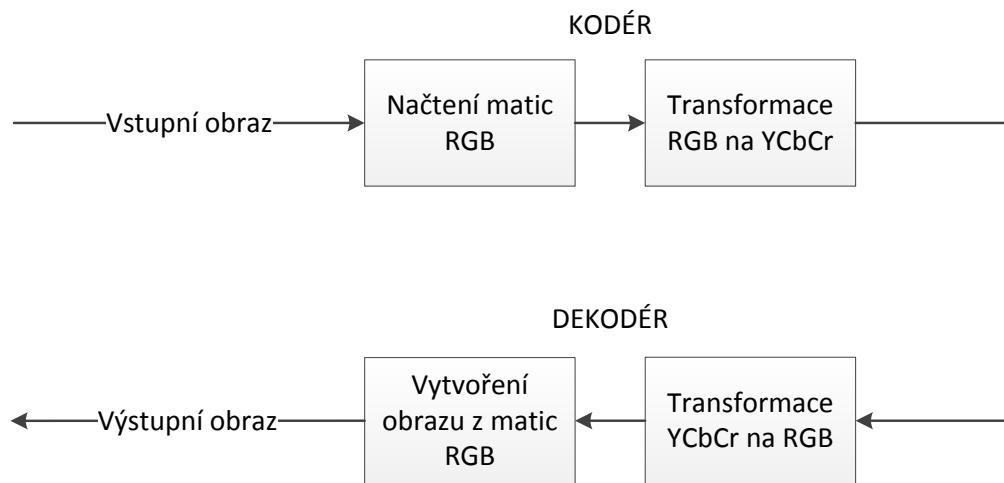
Obrázek 1.3: JFXMain

## 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

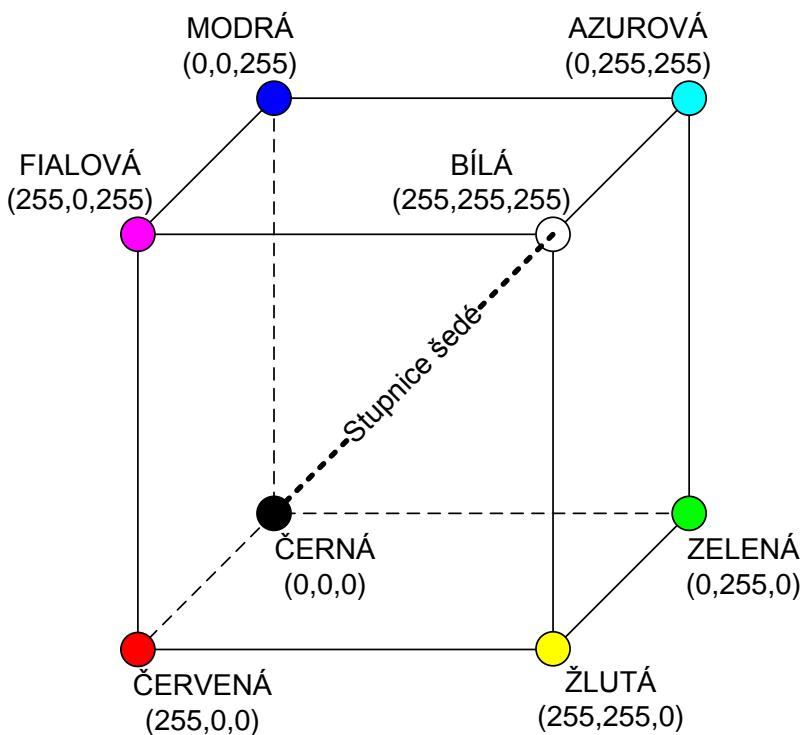
## 2.1 Teoretická část

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*

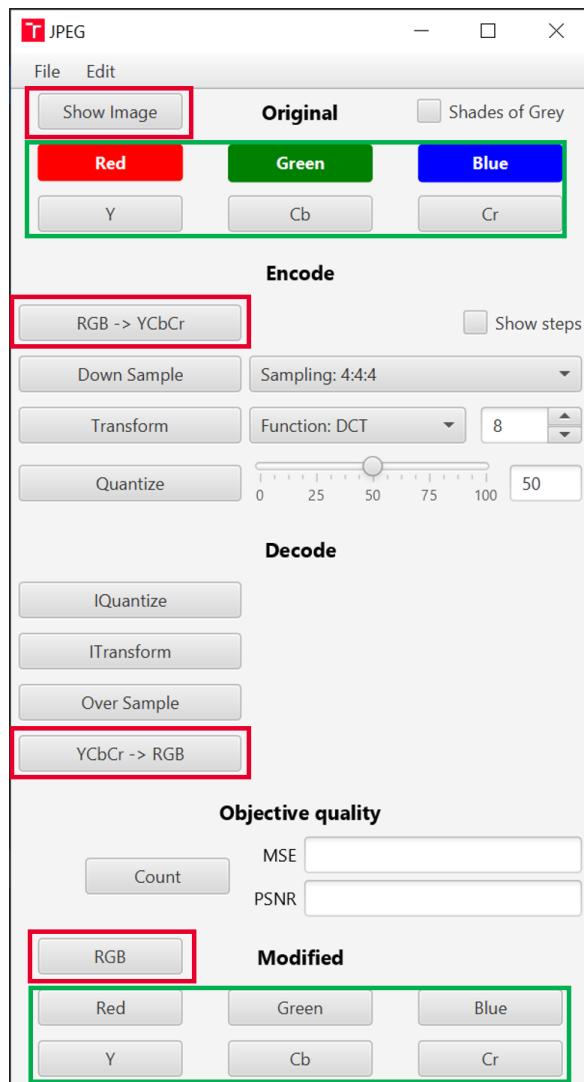
Model *RGB* je používán ve všech barevných zobrazovačích (televize, displeje apod.). Model není vhodný použít při kompresích standardy JPEG, MPEG apod. vzhledem k tomu, že při odstranění jakékoli informace lidské oko ihned rozpozná změnu obrazu. Z toho důvodu se při kompresích využívají modely oddělující 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

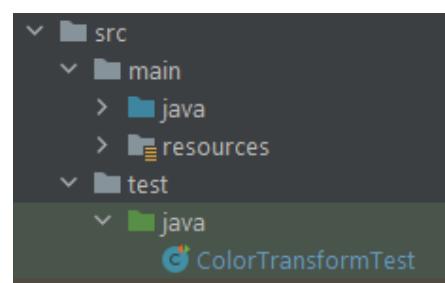
### 2.3.1 Na co si dát pozor?

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 spustíte 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 převodu RGB na YCbCr proběhl v pořádku
- Test převodu YcBcR na RGB proběhl v pořádku

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?

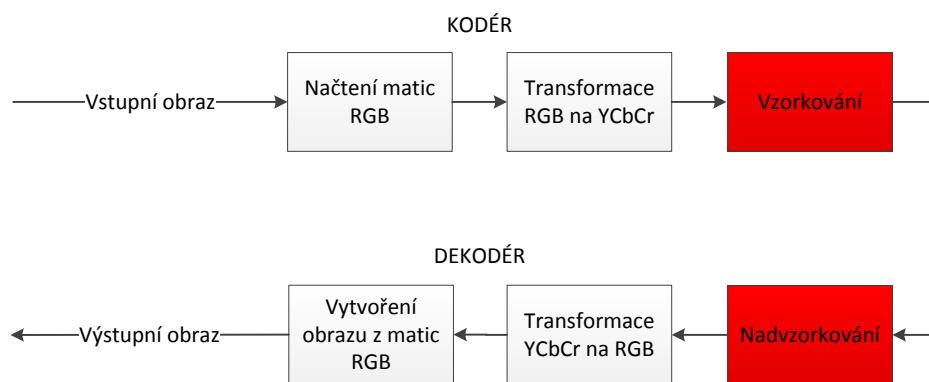
1. Vytvořte adresář, který se bude jmenovat jako Vaše ID.
2. Do adresáře nahrajte obsah vaší složky `src`.
3. Adresář komprimujte do ZIP a vložte ho do e-learningu.

## 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

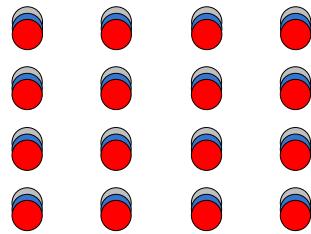
### 3.2 Teoretická část

#### Vzorkování modelu $YCbCr$

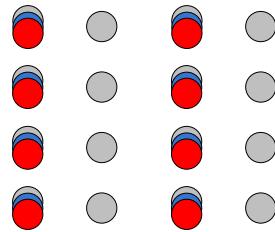
Model lidského zrakového systému HVS (*Human Visual System*) poukazuje na skutečnost, že lidské oko je méně citlivé na barevné složky než na jasové složky obrazu. Z toho důvodu je možné barvy jistým způsobem potlačit a to se děje vzorkováním barvonosných složek. Existují celkem 4 možnosti vzorkování: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

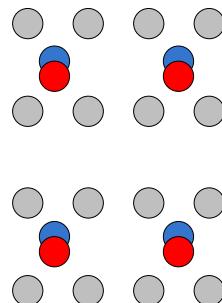
Při vzorkování dochází ke ztrátě informace.



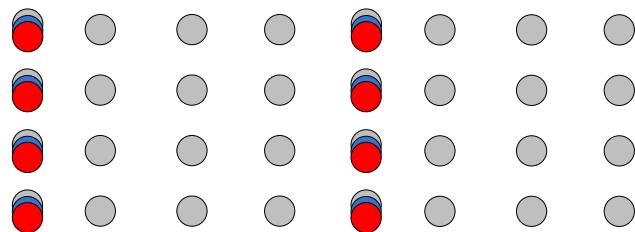
Obrázek 3.2: Model 4:4:4



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)

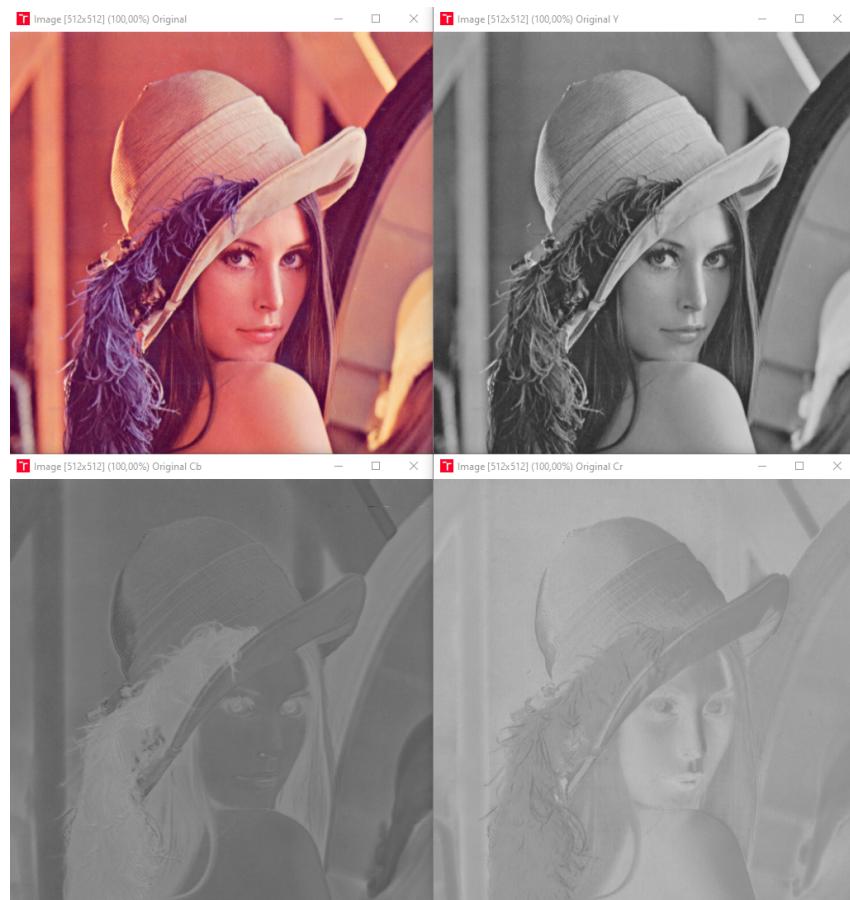
### 3.3.2 Podvzorkování modelu *YCbCr*

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.

Originální matice $O$	Podvzorkování	Podvzorkovaná matice $O_p$
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 3 5 7
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1
3 3 3 3 3 3 3 3	3 3 3 3 3 3 3 3	3 3 3 3
5 5 5 5 5 5 5 5	5 5 5 5 5 5 5 5	5 5 5 5
7 7 7 7 7 7 7 7	7 7 7 7 7 7 7 7	7 7 7 7
$O_p^T$	Podvzorkování	Podvzorkovaná matice $O_p^T$
1 1 1 1	1 3 5 7	1 1 1 1
3 3 3 3	1 3 5 7	3 3 3 3
5 5 5 5	1 3 5 7	5 5 5 5
7 7 7 7	1 3 5 7	7 7 7 7
Podvzorkovaná matice	Transponovaná $O_p^T$	Podvzorkování 4:2:0

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

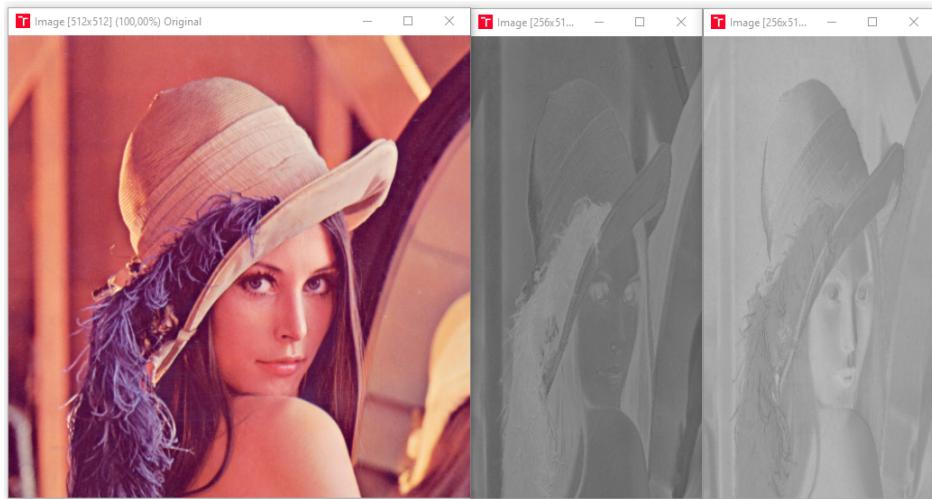
### 3.3.3 Nadvzorkování modelu YCbCr

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.

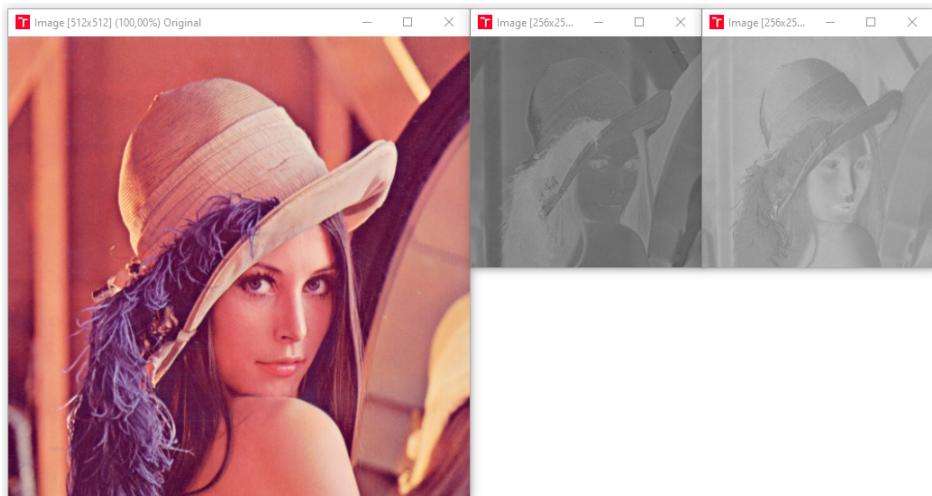
Výsledkem vaší práce by měli být funkční tlačítka pro vzorkování a nadvzorkování, včetně výběru typu vzorkování. Aplikace by také měla být schopná zobrazit vzorkované složky, aby bylo možné vizuálně ověřit, že metody skutečně fungují.

Výsledkem by měla být aplikace schopná zobrazit vzorkované výstupy pro všechny vzorkovací modely viz obrázky 3.12.

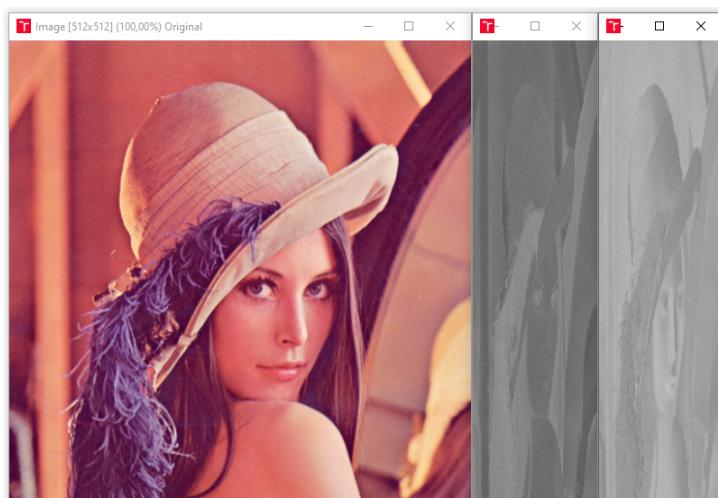
Třídu Sampling otestujete stejným postupem, tak jak je uvedeno v minulém cvičení 2.4, pouhým rozdílem bude v testovacím souboru, který se jmenuje `SamplingTest.java`.



(a) Podvzorkování 4:2:2



(b) Podvzorkování 4:2:0



(c) Podvzorkování 4:1:1

**Obrázek 3.12:** Vzorové obrázky při podvzorkování modelu  $YCbCr$