

Das Projekt *Image*

Ergänzungsfach Informatik

PD Dr. Victor Yakhontov

24. Februar 2022

Inhaltsverzeichnis

	Seite
1 Einführung	3
2 Lesen und Schreiben von Dateien mit Java	4
2.1 Klasse erstellen	4
2.2 Die main()-Methode	4
2.3 Variablen	5
2.4 Bild einlesen	5
2.5 Image in eine Bilddatei schreiben	6
2.6 Image aus einer Bilddatei einlesen und in eine Bilddatei schreiben. Vertiefung I	7
2.7 Image aus einer Bilddatei einlesen und in eine Bilddatei schreiben. Vertiefung II	10
2.8 Aufgaben zum Kapitel 2	15
3 Wie werden Pixelwerte in Java abgerufen und eingestellt	16
3.1 Aufgabe 1 zum Kapitel 3	16
RGB-Tabelle	16
3.2 Literale in Java (Vertiefung)	17
3.3 Abmessungen der 2D-Bilder	18
3.4 Darstellung von 2D-Bildpixeln	18
3.5 Voraussetzungen	19
3.6 Wie ermittelt man Bilddimensionen (Abmessungen)?	19
3.7 Aufgabe 2 zum Kapitel 3	20
3.8 Wie ermittelt man den Pixelwert?	20
3.9 Wie ermittelt man Alpha-, Red-, Green- und Blau-Werte des Pixels?	20
3.10 Aufgabe 3 zum Kapitel 3	21
3.11 Wie setzt man den Pixelwert?	22
3.12 Aufgabe 4 zum Kapitel 3:	22
4 Konvertierung eines Farnebildes in ein Graustufenbild	23
4.1 Voraussetzungen	23
4.2 Farbaufnahme zum Graustufenbild	23
4.3 Aufgabe zum Kapitel 4	23
5 Konvertierung eines Farnebildes in ein Negativenbild	24
5.1 Voraussetzungen	24
5.2 Farnebild zum Negativenbild	24
5.3 Aufgabe zum Kapitel 5	24

6	So erstellen Sie ein zufälliges Pixelbild in Java	25
6.1	Voraussetzungen	25
6.2	Farnenbild auf zufällige Pixelbild	25
6.3	Random-Pixel-Code	25
6.4	Generierung von zufälligen Pixeln	26
6.5	Bild schreiben	26
6.6	Aufgabe zum Kapitel 6	26
7	So erstellen Sie ein Spiegelbild in Java	27
7.1	Voraussetzungen	27
7.2	Spiegelbild erzeugen	27
7.3	Spiegelbildcode	28
7.4	Aufgaben zum Kapitel 7	28

1 Einführung

Das vorliegende Projekt widmet sich einigen Bildbearbeitungskonzepten. Sie werden diese Reise beginnen, indem Sie erfahren, wie man Image-Dateien mit Java einliest und schreibt. Danach wenden wir uns anderen Themen zu, wie man z.B. ein Farbbild in Graustufenbild umwandelt oder wie das Negativ eines Bildes erzeugt wird. Bei jedem Projektteil besprochen werden zunächst die grundlegende Theorie und danach die dazugehörigen Programmquellencodes. Alles was man für die Realisierung des Projektes braucht ist das Vorwissen über Image-Kodierung, Stellenwertsysteme von Zahlen sowie grundlegende Programmierkenntnisse. Java ist die Programmiersprache, welche für dieses Projekt hindurch verwendet wird.

- Extra Informationen oder zusätzliche Software erforderlich?
- Natürlich, Adobe Photoshop!
- Dies ist selbstverständlich nur ein Witz!
- Da dies ein *DO-IT-YOURSELF-Projekt* ist, werden wir nun alles von Grund auf neu errichten.

Zu beachten: Alle Textfragmente (im Gegenteil zu Quellcodefragmenten !), die in **Magenta** oder **Rot** vorkommen, sind "anklickbar" ! Durch einzelnen Mausklick auf solche Hyperlinks werden Sie je nach Kontext entweder zu einem anderen Textfragment (d.h. Referenzstellen) gebracht oder sogar zu referenzierten Internetseiten (auch zum Herunterladen !) automatisch weitergeleitet. Im letzteren Fall sollte dabei Ihr Standard-Internetbrowser automatisch gestartet und die verlinkte Internetseite geöffnet werden. Benutzen Sie unbedingt diese Möglichkeit beim Lesen und Navigieren im Text !

Viel Erfolg !

2 Lesen und Schreiben von Dateien mit Java

In diesem Projektteil werden wir lernen, wie man eine Bilddatei mittels der Java-Programmiersprache lesen und schreiben kann. Als Vorbereitung führen Sie die folgenden Schritte 1 bis 5 aus.

1. Starten Sie Teams oder Ihren Lieblings-Internetbrowser und laden Sie dort aus dem Verzeichnis **Projekte/Projekt_Image/** die Archivdatei Projekt_Image.zip in Ihr **Download**-Ordner herunter.
2. Entpacken Sie anschliessend diese Archivdatei direkt im Verzeichnis **../Eigene_Projekte/**. Dadurch wird darin ein neuer BlueJ-Ordner **../Eigene_Projekte/Projekt_Image** mit den Unterverzeichnissen **+libs** (**./flanagan.jar**, **./stdlib.jar**), **Images** (**./Doctor_Strange.png**, **./Sample_204_255_20_147.png**, **./Taj_Mahal3.png** und **./Taj_Mahal4.png**), **Docs** (**./Projektbeschreibung_und_Aufgaben.pdf**), **Leistungsnachweise** (**./***leer*****!) und einer Java-Klasse **./ImageInfo.java** angelegt.
3. Starten Sie jetzt BlueJ und öffnen Sie von dort aus das bereits vorhandene (und zur Zeit nur aus der einzelnen Java-Datei **ImageInfo.java** bestehende!) BlueJ-Projekt **../Eigene_Projekte/Projekt_Image**.
4. Legen Sie nun im Rahmen des Projektes routinemässig eine neue Klasse **MyImage.java** an.
5. Machen Sie nun die soeben erzeugte (Vorlage-)Datei **MyImage.java** im BlueJ-Editor auf und nehmen Sie daran die folgenden Änderungen vor.

Zum Lesen und Schreiben der Image-Datei müssen wir die File-Klasse importieren. Dazu schreiben wir:

```
import java.io. File ;
```

Wenn wir Schreib- und Leseoperationen durchführen, auch als I/O bzw. Eingabe/Ausgabe-Betrieb bekannt sind, können Fehler auftreten. Also, um Fehler zu behandeln, verwenden wir die **IOException**-Klasse:

```
import java.io. IOException;
```

Um das Bild im Speicher zu halten, erstellen wir (s. Kapitel 2.3) ein **BufferedImage**-Objekt und importieren dazu von Anfang an die **BufferedImage**-Klasse:

```
import java.awt.image. BufferedImage;
```

Um die Bildleseoperation durchzuführen, importieren wir auch die **ImageIO**-Klasse. Dazu schreiben wir:

```
import javax.imageio. ImageIO;
```

2.1 Klasse erstellen

Jetzt erstellen wir unsere **MyImage**-Klasse. Dazu schreiben wir:

```
public class MyImage { . . . }
```

Hinweis! Da unsere Dateinamen **MyImage.java** heisst, müssen unsere Klasse den Namen **MyImage** haben.

2.2 Die main()-Methode

Nun innerhalb dieser Klasse wird die Funktion **main()** definieren. Weil darin **I/O**-Operationen durchzuführen sind, muss die so genannte **IOException** direkt neben der Funktion **main()** geworfen werden. Unser Java-Code wird nun folgendermassen aussehen:

```
public class MyImage{  
    public static void main(String args[]) throws IOException{  
    }  
}
```

2.3 Variablen

Das Bild, das gelesen wird (s. unten), hat eine Breite von 1024px und eine Höhe von 768px. In einfachsten Fall müssen diese zwei Abmessungen **im Voraus (d. h. vor dem Einlesen der Bilddatei !)** bekannt sein. Wir deklarieren also zwei Variablen innerhalb der `main()`-Funktion, um die Dimension des Bildes festzulegen:

```
int width = 1024; // Image-Breite (image width)
int height = 768; // Image-Höhe (image height)
```

Als Nächstes erstellen wir ein `BufferedImage`-Variablenbild und die `File`-Dateivariablen `f` (d.h. der Name der Bilddatei) und setzen sie beide auf `null`:

```
BufferedImage image = null;
File f = null;
```

Nun sieht unser Code so aus:

```
public class MyImage {
    public static void main(String args[]) throws IOException {
        int width = 1024; // width of the image
        int height = 768; // height of the image
        BufferedImage image = null;
        File f = null;
    }
}
```

2.4 Bild einlesen

Nun werden wir die Bilddatei lesen. Beachten Sie, dass es während der Lese/Schreib-Operation immer vorgeschlagen wird, den Try-catch-Block zu verwenden. Dies liegt daran, dass die IO-Operationen Ausnahme (Fehler) generieren können. Um die IO-Ausnahmen behandeln zu können, benötigen wir einen Ausnahmebehandlungscode. Wenn wir den Try-catch-Block nicht verwenden, stürzt unser Code einfach ab, wenn eine Ausnahme (Fehler) auftritt, wenn z.B. die einzulesende Bilddatei **nicht gefunden wird**. Wir schreiben also

```
try{
    // ein Java-Code kommt her ...
} catch (IOException e) {
    // ein Java-Code kommt her ...
}
```

Innerhalb des try-Blocks werden wir ein Objekt der `File`-Klasse erstellen und ihm als Parameter den Image-Dateipfad übergeben:

```
f = new File("Images/Taj_Mahal3.png"); // Image-Dateipfad
```

Hinweis! Die Bilddatei, die wir in der obigen Zeile verwenden, heisst `Taj_Mahal3.png` und befindet sich im `Images`-Ordner. Überprüfen Sie mit Hilfe eines gewöhnlichen Windows-Programms die Breite (= 1024px) und die Höhe (= 768px) des Bildes in Pixeln, die in der obigen Zeile verwendet werden.

Als Nächstes erstellen wir ein Objekt vom Typ `BufferedImage` und übergeben als Parameter die Werte von `width`, `height` sowie den `int`-Typ des Images.

```
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
```

Hinweis! `TYPE_INT_ARGB` bedeutet, dass wir die Alpha-, Rot-, Grün- und Blaukomponente des Bildpixels mit jeweils 8-Bit-Integer-Wert darstellen.

Als nächstes lesen wir das Bild mit der Funktion `read()` der `ImageIO`-Klasse ein und übergeben ihr als Parameter den Bilddateipfad, den wir in der Variablen `f` gespeichert haben.

```
image = ImageIO.read(f);
```

Falls das Einlesen erfolgreich abläuft, geben wir die Meldung "Read Complete." auf die Konsole aus:

```
System.out.println("Reading complete.");
```

Sonst geben wir im Inneren des try-catch-Blocks die Fehlermeldung auf die Konsole aus:

```
System.out.println("Error: " + e);
```

Nun sieht unser Code so aus:

```
1 import java.io. File ;
2 import java.io. IOException;
3 import java.awt.image. BufferedImage;
4 import javax.imageio. ImageIO;
5 public class MyImage{
6     public static void main(String args[]) throws IOException {
7         int width = 1024; // width of the image
8         int height = 768; // height of the image
9         BufferedImage image = null;
10        File f = null;
11
12        // read image
13        try {
14            f = new File("Images/Taj_Mahal3.png"); // image file path
15            image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
16            image = ImageIO.read(f);
17            System.out.println("Reading complete.");
18        } // end try
19        catch(IOException e) {
20            System.out.println("Error: " + e); } // end catch
21        } // end main
22    } // end class MyImage
```

Java-Programm zum Einlesen aus einer Bilddatei

2.5 Image in eine Bilddatei schreiben

Um das Bild als Datei zu schreiben, verwenden wir den try-catch-Block. Wir erstellen zunächst ein Objekt vom Dateityp File und übergeben als Parameter den **neuen Bilddateipfad**, in den wir das bereits eingelesene Bild schreiben (speichern) wollen. Dazu schreiben wir:

```
f = new File("Images/Taj_Mahal3.jpg"); // output file path
```

Als nächstes schreiben wir das Bild image mit der write()-Funktion der ImageIO-Klasse:

```
ImageIO.write(image, "jpg", f);
```

Hinweis! Der Parameter image hierher ist die Variable, welche mit dem zu speichernden Bild belegt wird; "jpg" ist die Zieldateiformat, das sich von unserem ursprünglichen "png"-Format unterscheiden darf; f ist die Variable, welche mit dem (neuen) Zieldateipfadnamen zu belegen ist. Anders gesagt, unser Programm wandelt also im einfachsten Fall das Bild aus dem ursprünglichen png-Format ins jpg-Format um.

Dabei wird "Writing Complete." auf die Konsole ausgegeben:

```
System.out.println("Writing complete.");
```

Im Inneren des try-catch-Blocks geben wir die Fehlermeldung aus. Dazu schreiben wir:

```
System.out.println("Error: " + e);
```

Unser endgültiger Code sieht nun so aus:

```
1 import java.io. File ;
2 import java.io. IOException;
3 import java.awt.image. BufferedImage;
4 import javax.imageio. ImageIO;
5 public class MyImage{
6     public static void main(String args[]) throws IOException {
7         int width = 1024;    // width of the image
8         int height = 768;    // height of the image
9         BufferedImage image = null;
10        File f = null;
11
12        // read image
13        try {
14            f = new File("Images/Taj_Mahal3.png"); // image file path
15            image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
16            image = ImageIO.read(f);
17            System.out.println("Reading complete.");
18        } //end try
19        catch(IOException e) {
20            System.out.println("Error: " + e); } // end catch
21
22        // write image
23        try {
24            f = new File("Images/Taj_Mahal3.jpg"); // output file path
25            ImageIO.write(image, "jpg", f);
26            System.out.println("Writing complete.");
27        } catch(IOException e){
28            System.out.println("Error: "+e); } // end catch
29    } // end main
30 } // end MyImage
```

Programmausdruck 1: Programm zum Einlesen aus einer Bilddatei und zum Schreiben in eine.

Sie können einen der beiden Codes zum Lesen und Schreiben von Bilddateien in Java verwenden.

2.6 Image aus einer Bilddatei einlesen und in eine Bilddatei schreiben. Vertiefung I

Sollten die Breite (width) und die Höhe (height) einer einzulesenden Bilddatei (z.B. Images/Taj_Mahal3.png) *a priori*, d.h. **vor dem Einlesen, unbekannt sein**, muss der Programm Quellcode 1 modifiziert werden. Dafür werden (s. Programm Quellcode 2) zwei statt einer (!) Variablen, `image_on_disk` und `image_memory`, vom Datentyp `BufferedImage` deklariert. Die erste davon, `image_on_disk`, wird mittels des Einlesens der Bilddatei initialisiert und dadurch zunächst für die Ermittlung der Abmessungen des Bildes verwendet. Mit deren Hilfe sowie mittels `image_on_disk` wird anschliessend das Output-Bild kreiert und somit die 2. Variable `image_memory` belegt. Für die Erstellung der Bildes anhand `BufferedImage` ist Folgendes zu beachten.

Alle Bilddateien im png-Format, welches ständig fürs ganze Projekt zum Einsatz kommt, können zusätzlich zu den RGB-Farben die so genannte **Transparenzinformationen** enthalten (s. Projektteil 3). Die letztere wird für jedes Pixel in Form eines Alpha-Werts $\alpha = 0, 1, \dots, 255$ des **Alphakanals** in der Bilddatei integriert und gespeichert. Der α -Wert gibt nämlich eine mathematisch exakte Information für jedes einzelne Pixel an, wie viel Prozent (0% bis 100%) dieses Pixels vom Hintergrund des Bildes durchsichtig sein soll: $\alpha = 0$ bzw. $\alpha = 255$ entspricht voller Durchsichtigkeit (oder Transparenz) bzw. voller Undurchsichtigkeit (Intransparenz oder Opazität) des Pixels. Diese zwei Extremsituationen werden zum Vergleich in Abb. 1 und 2 dargestellt.

Das png-Format unterstützt Alphakanäle mit 8 Bits, was $2^8 = 256$ Abstufungen der Transparenzstärke entspricht, und erlaubt somit, und zwar unabhängig von der Hintergrundfarbe, die Kanten von Text und Bildern zu glätten. Man kann echte Schlagschatten verwenden, die im Hintergrund ausblenden, oder Bilder erzeugen, die beliebig geformt erscheinen – wenn das Anzeigeprogramm das png-Format beherrscht.

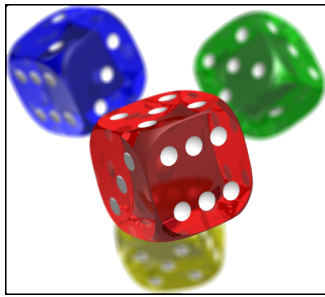


Abb. 1: Die sämtlichen violetten Pixel dieses Bildes (vgl. mit Abb. rechts) sind völlig transparent (durchsichtig) und entsprechen dem Alpha-Wert $\alpha = 0$, so dass man dadurch den weissen Hintergrund sieht.

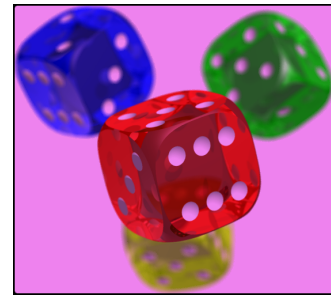


Abb. 2: Die sämtlichen violetten Pixel dieses Bildes sind völlig intransparent (undurchsichtig) und entsprechen dem Alpha-Wert $\alpha = 255$, so dass man dadurch **keinen** weissen Hintergrund sieht.

Die oben aufgeführten wichtigen Transparenz-Eigenschaften von png-Bilddateien unterscheiden sich von denjenigen der jpg-Bilddateien, welche unter anderem **keine Transparenzinformation enthält**. Sollte also eine png-Bilddatei mit 4 ARGB-Kanälen in eine Bilddatei im jpg-Format (d.h. mit nur 3 Kanälen) konvertiert werden, braucht der Alphakanal eine Spezialbehandlung. Die letztere wird im unteren Programmquellcode 2 realisiert, in dem wir bei `image_memory` zunächst mittels eines `createGraphics().drawImage`-Befehls einen **weissen Hintergrund** (kann natürlich durch einen schwarzen, blauen, etc. ersetzt werden) kreieren und darauf das bereits in der `BufferedImage`-Variablen `image_on_disk` gespeicherte Bild aufmalen.



Achtung !

Damit Sie das unten stehende Programm "MyImage_ohne_A_mit_A" kompilieren und zum Laufen bringen könnten, muss im Ordner `+libs` des Projektverzeichnis die `flanagan.jar`-Bibliothek vorhanden sein !

```

1 import java.awt.Color;
2 import java.awt.image.BufferedImage;
3 import java.io. File ;
4 import java.io. IOException;
5 import javax.imageio.ImageIO;
6 import flanagan.io.Db;
7 import flanagan.io.FileNameSelector;
8 import java.util. regex. Pattern;
9
10 public class MyImage_ohne_A_mit_A {
11
12     public static void main(String [] args) {
13
14         boolean answer = false, file_exist = false;
15         String inFileName, inFileNamePfad, outFileName,
16             outFileNamePfad, outFormat = "";
17
18         FileNameSelector fc = new FileNameSelector("Images/");
19         File fin = null, fout = null;
20
21         while (!answer && !file_exist) {
22             inFileName = fc.selectFile ("Waehle das Input-File:");
23             inFileNamePfad = "Images/" + inFileName;
24
25             fin = new File(inFileNamePfad); // input file path
26
27             // Ermitteln den Stamm- und die Erweiterung-Teile von inFileName:

```



```

28 String [] inFileName_Teil = inFileName.split(Pattern.quote(" "));
29 // inFileName_Teil[0] = Stammname, inFileName_Teil[1] = Erweiterung (png, jpg, ...)
30
31 outFileName = Db.readLine("Geben_Sie_den_Namen_des_Output-Files_ +
32 "MIT_einer_Erweiterung_(png,_jpg,_etc.)_ein" + "\n" +
33 "und_druecken_die_ENTER-Taste_fuer_Bestaetigung!",
34 inFileName_Teil[0] + ".jpg");
35
36 // Ermitteln den Stamm- und die Erweiterung-Teile von outFileName:
37 String [] outFileName_Teil = outFileName.split(Pattern.quote(" "));
38 // outFileName_Teil[0] = Stammname, outFileName_Teil[1] = Erweiterung (png, jpg, ...)
39
40 outFileNamePfad = "Images/" + outFileName;
41
42 outFormat = outFileName_Teil[1].trim();
43
44 fout = new File(outFileNamePfad); // output file path
45
46 if (inFileNamePfad.equals(outFileNamePfad)) {
47     file_exist = Db.noYes("Die_Input-_und_Output-Bilddateien_ +
48 "haben_die_gleichen_Namen!" + "\n" +
49 "Moechten_Sie_trotzdem_fortsetzen?");
50
51     if (! file_exist ) {
52         // file_exist = false;
53         System.out.println("file_exist=_ " + file_exist +
54 " _answer=_ " + answer);
55         continue;
56     }
57 }
58
59 answer = Db.yesNo("Die_Angaben_lauten:" + "\n" +
60 "Input-Bilddatei:_ " + "\"" + inFileNamePfad + "\"" + "\n" +
61 "Output-Bilddatei:_ " + "\"" + outFileNamePfad + "\"" + "\n" +
62 "Die_Erweiterung_des_Input-Files:_ " + "\"" +
63 inFileName_Teil[1] + "\"" + "\n" +
64 "Die_Erweiterung_des_Output-Files:_ " + "\"" +
65 outFileName_Teil[1] + "\"" + "\n\n" +
66 "Sind_Sie_damit_einverstanden?");
67 }
68
69 BufferedImage image_on_disk = null, image_memory = null;
70 boolean Alpha;
71
72 //***** Image einlesen *****
73
74 try {
75     image_on_disk = ImageIO.read(fin);
76     System.out.println("Reading_ " + fin + "_complete_!"); } // end try
77 catch(IOException e){
78     System.out.println("Error:_ " + e); } // end catch
79
80 int width = image_on_disk.getWidth(); // Breite des Images
81 int height = image_on_disk.getHeight(); // Höhe des Images
82
83 System.out.println("\n" + "width=_ " + width + " _height=_ " + height + "\n");

```

```

84
85     if (image_on_disk.getColorModel().hasAlpha()) {
86         Alpha = true;
87         System.out.println("Die_Datei_HAT_Alpha-Kanal\n");
88         // image_on_disk hat A-Kanal
89     } else {
90         Alpha = false;
91         System.out.println("Die_Datei_hat_KEINEN_Alpha-Kanal\n");
92         // image_on_disk hat keinen A-Kanal
93     }
94
95     //***** Image kreieren und anpassen *****
96
97     if (outFormat.equals("png") && Alpha) { // png4 <-> png4
98         System.out.println("Wir_sind_da:_1\n");
99         image_memory = new BufferedImage(width, height,
100             BufferedImage.TYPE_INT_ARGB);
101         // Typ TYPE_INT_ARGB, 4 Kanäle: ARGB, d.h. MIT Alpha-Kanal
102
103         // Kreieren das GLEICHE 4-Kanal Image wie image_on_disk und mit
104         // Aufrechtserhaltung des Alpha-Kanals
105         image_memory.createGraphics().drawImage(image_on_disk, 0, 0, null);
106     }
107     else { // png4/3 -> jpg oder png3 -> png3 oder jpg -> png3
108         System.out.println("Wir_sind_da:_2\n");
109         image_memory = new BufferedImage(width, height,
110             BufferedImage.TYPE_INT_RGB);
111         // Typ TYPE_INT_RGB, 3 Kanäle: RGB, d.h. OHNE Alpha-Kanal
112
113         // Kreieren ein Image mit der gleichen Breite and Höhe wie image_on_disk und
114         // WEISSEM Hintergrund zum Ersetzen des A-Kanal
115         image_memory.createGraphics().drawImage(image_on_disk, 0, 0,
116             Color.WHITE, null);
117     }
118
119     //***** Image schreiben *****
120
121     try {
122         ImageIO.write(image_memory, outFormat, fout);
123         System.out.println("Writing_" + fout + "_complete!"); } // end try
124     catch(IOException e) {
125         System.out.println("Error:_ " + e); } // end catch
126
127     System.exit(1); // Programm beenden
128
129 } // end main
130 } // end class MyImage_ohne_A_mit_A

```

Programmausdruck 2: Klasse MyImage_ohne_A_mit_A.java: Java-Programm zum Einlesen aus einer Bilddatei und zum Schreiben in eine (neue) Bilddatei ohne Vorwissen über ihre Abmessungen und mit Spezialbehandlung des Alphakanals.

2.7 Image aus einer Bilddatei einlesen und in eine Bilddatei schreiben. Vertiefung II

Eine alternative und etwas effizienterer Methode, um die Breite und die Höhe der einzulesenden Bilddatei zu ermitteln, besteht darin, dass man **vor dem Einlesen des Bildfiles** (z.B. Images/Taj_Mahal3.png) eine exter-

ne Klasse `ImageInfo.java` aufruft (s. die Klasse `ImageInfo` 4 unten). Diese Hilfsklasse liest nämlich nicht die ganze png-Bilddatei ein, sondern nur einen kleinen Teil (wenige Bytes) davon, der unter anderem den Filetyp (png, jpg, etc.) sowie die Bildabmessungen enthält. Diese Eigenschaften bzw. Parameter werden schlussendlich auf die Konsole ausgegeben. Sollte die Bilddateigröße mehrere Megabytes ausmachen, bietet dieses Verfahren eine deutlich höhere Speichereffizienz gegenüber der vorigen Methode an und eignet sich deshalb vor allem zum Einlesen und Wiederspeichern von png-Bilddateien mit 4 ARGB-Kanälen, da der Einfachheit halber **keine Spezialbehandlung von Alpha-Werten** (s. Programm 2 für eine solche Behandlung!) darin eingebaut wird.



Achtung !

Damit das unten stehende Programm `MyImageInfo.java` kompiliert und zum Laufen gebracht werden könnte, muss im Projektverzeichnis die Java-Klasse `ImageInfo.java` vorliegen.

Die auf dieser Weise abgeänderte Version der obigen Klasse `MyImage.java` (1) heisst nun `MyImageInfo.java` und sieht wie folgt aus:

```
1 import java.io. File ;
2 import java.io. IOException;
3 import java.awt.image. BufferedImage;
4 import javax.imageio. ImageIO;
5
6 public class MyImageInfo{
7     public static void main(String args[]) throws IOException{
8         BufferedImage image_on_disk = null;
9         File fin = new File("Images/Taj_Mahal3.png"), //input file path
10        fout = new File("Images/Taj_Mahal3.jpg"); //output file path
11        // Aufruf einer EXTERNEN Klasse ImageInfo.java, welche bereits VOR der Kompilierung
12        // von MyImageInfo im gleichem Projektverzeichnis vorhanden sein soll !
13        ImageInfo imageInfo = new ImageInfo(fin);
14        System.out.println("Informationen zum input-File " + fin + ":\n" + imageInfo + "\n");
15        int width = imageInfo.width; // Zuweisung von width = width of the image
16        int height = imageInfo.height; // Zuweisung von height = height of the image
17
18        image_on_disk = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
19
20        //read image
21        try {
22            image_on_disk = ImageIO.read(fin);
23            System.out.println("Reading " + fin + " complete!"); } // end try
24        catch(IOException e){
25            System.out.println("Error: " + e); } // end catch
26
27        //write image
28        try {
29            ImageIO.write(image_on_disk, "jpg", fout);
30            System.out.println("Writing " + fout + " complete!"); } // end try
31        catch(IOException e) {
32            System.out.println("Error: " + e); } // end catch
33    } // end main()
34 } // end MyImageInfo
```

Programmausdruck 3: Klasse `MyImageInfo.java`: Java-Programm zum Einlesen aus einer Bilddatei und zum Schreiben in eine (neue) Bilddatei ohne Vorwissen über ihre Abmessungen.

Wie bereits oben erklärt wurde, ruft die obige Klasse `MyImageInfo.java` eine zweite Klasse `ImageInfo.java` auf, um die erforderlichen Informationen über den Typ, die Breite und die Höhe des Bildes direkt aus der Bilddatei zu ermitteln. Der Programmquellcode der Klasse `ImageInfo.java` ist unten präsentiert:

```
1 import java.io.ByteArrayInputStream;
2 import java.io.File;
3 import java.io.FileInputStream;
4 import java.io.IOException;
5 import java.io.InputStream;
6
7 public class ImageInfo {
8     int height, width;
9     String mimeType;
10
11     private ImageInfo() { }
12
13     public ImageInfo(File file) throws IOException {
14
15         FileInputStream is = null;
16
17         try {
18             is = new FileInputStream(file);
19             processStream(is);
20         }
21         catch(IOException e){
22             System.out.println("Error: " + e + "\n" + "Programmist sofort beendet!");
23             System.exit(0);
24         }
25         finally {
26             try {
27                 if (is != null) is.close();
28             } catch (IOException e) {
29                 System.out.println("Error: " + e + "\n" + "Programmist sofort beendet!");
30                 System.exit(0);
31             }
32         }
33     }
34
35     public ImageInfo(InputStream is) throws IOException {
36         processStream(is);
37     }
38
39     public ImageInfo(byte[] bytes) throws IOException {
40         InputStream is = new ByteArrayInputStream(bytes);
41         try {
42             processStream(is);
43         } finally {
44             is.close();
45         }
46
47     private void processStream(InputStream is) throws IOException {
48         int c1 = is.read();
49         int c2 = is.read();
50         int c3 = is.read();
51
52         mimeType = null;
```

```

53 width = height = -1;
54 if (c1 == 'G' && c2 == 'I' && c3 == 'F') { // GIF
55     is.skip(3);
56     width = readInt(is, 2, false);
57     height = readInt(is, 2, false);
58     mimeType = "image/gif";
59 } else if (c1 == 0xFF && c2 == 0xD8) { // JPG
60     while (c3 == 255) {
61         int marker = is.read();
62         int len = readInt(is, 2, true);
63         if (marker == 192 || marker == 193 || marker == 194) {
64             is.skip(1);
65             height = readInt(is, 2, true);
66             width = readInt(is, 2, true);
67             mimeType = "image/jpeg";
68             break;
69         }
70         is.skip(len - 2);
71         c3 = is.read();
72     }
73 } else if (c1 == 137 && c2 == 80 && c3 == 78) { // PNG
74     is.skip(15);
75     width = readInt(is, 2, true);
76     is.skip(2);
77     height = readInt(is, 2, true);
78     mimeType = "image/png";
79 } else if (c1 == 66 && c2 == 77) { // BMP
80     is.skip(15);
81     width = readInt(is, 2, false);
82     is.skip(2);
83     height = readInt(is, 2, false);
84     mimeType = "image/bmp";
85 } else {
86     int c4 = is.read();
87     if ((c1 == 'M' && c2 == 'M' && c3 == 0 && c4 == 42)
88         || (c1 == 'I' && c2 == 'I' && c3 == 42 && c4 == 0)) { // TIFF
89         boolean bigEndian = c1 == 'M';
90         int ifd = 0;
91         int entries;
92         ifd = readInt(is, 4, bigEndian);
93         is.skip(ifd - 8);
94         entries = readInt(is, 2, bigEndian);
95         for (int i = 1; i <= entries; i++) {
96             int tag = readInt(is, 2, bigEndian);
97             int fieldType = readInt(is, 2, bigEndian);
98             long count = readInt(is, 4, bigEndian);
99             int valOffset;
100             if ((fieldType == 3 || fieldType == 8)) {
101                 valOffset = readInt(is, 2, bigEndian);
102                 is.skip(2);
103             } else {
104                 valOffset = readInt(is, 4, bigEndian);
105             }
106             if (tag == 256) {
107                 width = valOffset;
108             } else if (tag == 257) {

```

```

109         height = valOffset; }
110     if (width != -1 && height != -1) {
111         mimeType = "image/tiff";
112         break;
113     }
114 }
115 }
116 }
117 if (mimeType == null) {
118     throw new IOException("Unsupported image type");
119 }
120 }
121
122 private int readInt(InputStream is, int noOfBytes, boolean bigEndian) throws IOException {
123     int ret = 0;
124     int sv = bigEndian ? ((noOfBytes - 1) * 8) : 0;
125     int cnt = bigEndian ? -8 : 8;
126     for(int i=0;i<noOfBytes;i++) {
127         ret |= is.read() << sv;
128         sv += cnt;
129     }
130     return ret;
131 }
132
133 public int getHeight() {
134     return height;
135 }
136
137 public void setHeight(int height) {
138     this.height = height;
139 }
140
141 public int getWidth() {
142     return width;
143 }
144
145 public void setWidth(int width) {
146     this.width = width;
147 }
148
149 public String getMimeType() {
150     return mimeType;
151 }
152
153 public void setMimeType(String mimeType) {
154     this.mimeType = mimeType;
155 }
156
157 @Override
158 public String toString() {
159     return "MIME_Type: " + mimeType + "\tWidth: " + width +
160         "\tHeight: " + height; }
161 }

```

Programmausdruck 4: Klasse ImageInfo.java: Eine Hilfsklasse zur Bestimmung des Typs sowie der Abmessungen einer Bilddatei.


2.8 Aufgaben zum Kapitel 2

1. Erstellen und kompilieren Sie die oben erwähnten Klassen

`MyImage.java` (1), `MyImage_ohne_A_mit_A` (2), `MyImageInfo.java` (3)

und führen Sie anschliessend alle drei Versionen des Programms aus.

2. Laden Sie aus dem Internet weitere beliebige png- bzw. jpg-Bilddateien herunter und speichern sie ebenfalls ins Unterverzeichnis Images des Projektverzeichnisses.
3. Nehmen Sie erforderliche Änderungen bei jedem der drei vorliegenden Programme vor und führen Sie die png \rightarrow png, png \rightarrow jpg und jpg \rightarrow png Konvertierung ihrer eigenen Bilddateien durch.
4. Vergewissern Sie sich, dass bei jedem Programmablauf eine neue Bilddatei mit den von Ihnen vordefinierten Namen und Erweiterung (png bzw. jpg) im Image-Ordner des Projektverzeichnisses entsteht.
5. Untersuchen Sie die Programmstruktur der Klasse `MyImage_ohne_A_mit_A` (2) und erstellen Sie dafür, das entsprechende Programmablaufdiagramm mit Hilfe des **draw**-Programms, welches Ihnen frei zur Verfügung steht. Exportieren Sie das PA-Diagramm aus dem draw-Programm in eine pdf-Datei, welche in den Ordner Images des Projektverzeichnisses unter dem Namen `PAP_MyImage_ohne_A_mit_A.pdf` zu speichern ist. Legen Sie Ihrer Lösung den Ausdruck dieses Files bei.
6. Nehmen Sie notwendige Änderungen beim Programm `MyImage_ohne_A_mit_A` (2) vor, damit eine neue png-Bilddatei nicht mit einer weissen, sondern mit der Hintergrundfarbe Ihrer Wahl (s. Tabelle 1) entsteht.
7. Experimentieren Sie mit verschiedenen png-Bilddateien und diversen Hintergründen. Achten Sie jeweils darauf, *welcher Bilddateityp vorliegt und ob der Alphakanal im Bild eingebettet ist*.
8. Untersuchen Sie jeweils die Eigenschaften der Bilddateien z.B. mit Hilfe des **GIMP-Programms**.

Hinweis: Dabei benutzen Sie das Instrument Farbpipette. Dieses lässt sich entweder über das Menü Werkzeuge \rightarrow Farbpipette im Bildfenster oder durch einen Mausklick auf das Symbol  im Werkzeugkasten wählen. Der Mauszeiger nimmt dann die Form einer Pipette an, womit Sie durch Anklicken mit der linken Maustaste an verschiedenen Bildstellen die entsprechende Vordergrundfarbe aufnehmen können. Durch Umschalt-Klick (ggf. Shift-Klick) wird dabei ein extra Info-Fenster geöffnet, wo Sie auch detaillierte Informationen über die einzelnen Farbkomponenten des Pixels sowie dessen Alphakanals erhalten.

3 Wie werden Pixelwerte in Java abgerufen und eingestellt

Dieser Theorieteil des Image-Projektes widmet sich der Pixelstruktur eines Bildes bzw. einer Bilddatei. Im Einzelnen wird es genau untersucht, Pixelwerte in einer Variablen zu speichern, um sie in einem Java-Programm einzustellen und zu erhalten.

Die kleinste Einheit eines Bildes wird Pixel genannt und besteht im Allgemeinen aus 4 Komponenten:

- Alpha
- Rot (= Red)
- Grün (= Green)
- Blau (= Blue)



Beachte:

1. Die Alpha-Komponente bestimmt die Transparenz, während Rot, Grün und Blau (genau in dieser Reihenfolge!) die Farbe des Pixels definieren. Im allgemeinen bezeichnen wir diese vier Komponenten als A für Alpha, R für Rot, G für Grün und B für Blau.
2. Jede dieser vier Komponenten (ARGB) nimmt einen Wert zwischen 0 bis 255 einschliesslich, wobei 0 bedeutet, dass die Komponente fehlt und 255 bedeutet, dass die Komponente vollständig vorhanden ist.

Da $2^8 = 256$ ist, so können 8 Bits einen Wert im Bereich von 0 bis 255 darstellen. Das heisst, dass wir nur 8 Bits benötigt, um den Wert von jedem der vier Komponenten zu speichern. Weil ein Pixel vier Komponenten aufweist, ist die Gesamtzahl der Bits, die erforderlich ist, um den Wert aller vier Komponenten ARGB zu speichern, gleich $4 \times 8 = 32$ Bits oder 4 Bytes.

Das unten gezeigte Bild stellt einen einzigen Pixelwert dar, der aus 32 Bits besteht. Alpha nehmen die am weitesten links liegenden 8 Bits, während Blue die rechten 8 Bits des Pixels annimmt. Das erste Bit befindet sich an der rechten Seite und ist nummeriert oder indexiert mit 0 und das letzte Bit befindet sich an der äussersten linken Seite und ist nummeriert oder indexiert mit 31.

ALPHA								RED								GREEN								BLUE							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Jede dieser Bitpositionen nimmt einen der beiden Bitwerte oder binären Werte an, d.h. 0 oder 1. Wenn alle 8 Bits von ALPHA auf 0 gesetzt sind, d.h. 0000 0000, wird das Pixel völlig transparent (oder durchsichtig) und wird daher als **farblos wahrgenommen**. Man sieht also nur einen farblosen Hintergrund und zwar unabhängig von eingestellten RGB-Werten. Nach der Dezimal-Umwandlung erhalten wir in diesem Falle also $0000\,0000_2 = 0_{10}$. In ähnlicher Weise, wenn alle 8 Bits von ALPHA auf 1 gesetzt sind, d.h. 1111 1111, dann erhält man: $1111\,1111_2 = 255_{10}$, was einem **vollkommen undurchsichtigen Pixel** entspricht, so dass die RGB-Pixelfarbe in vollem Umfang wahrgenommen wird.

Hinweis! Wir können die Bitwerte auch in *hexadezimaler Form* darstellen. So $0000\,0000_2$ in binärer ist gleich 00_{16} in hexadezimaler Darstellung. In ähnlicher Weise gilt: $1111\,1111_2 = FF_{16} = 255_{10}$.

3.1 Aufgabe 1 zum Kapitel 3

Unter dem Link <http://www.tydac.ch/color/> finden Sie einen komfortablen RGB-Color-Calculator (RGB-Farbenrechner), welcher es ermöglicht, jegliche Farbe durch direkte Eingabe der einzelnen RGB-Werte darzustellen. Prüfen Sie mit dessen Hilfe einige der in der unten stehenden Tabelle angegebenen RGB-Tripel nach.

aliceblue (240, 248, 255)	antiquewhite (250, 235, 215)	aqua (0, 255, 255)	aquamarine (127, 255, 212)	azure (240, 255, 255)	beige (245, 245, 220)	bisque (255, 228, 196)
black (0, 0, 0)	blanchedalmond (255, 235, 205)	blue (0, 0, 255)	blueviolet (138, 43, 228)	brown (165, 42, 42)	burlywood (222, 184, 135)	cadetblue (95, 158, 160)
chartreuse (127, 255, 0)	chocolate (210, 105, 30)	coral (255, 127, 80)	cornflowerblue (100, 149, 237)	cornsilk (255, 248, 220)	crimson (220, 20, 60)	cyan (0, 255, 255)
darkblue (0, 0, 139)	darkcyan (0, 0, 139, 139)	darkgoldenrod (184, 134, 11)	darkgray (169, 169, 169)	darkgreen (0, 100, 0)	darkkhaki (189, 183, 107)	darkmagenta (139, 0, 139)
darkolivegreen (85, 107, 47)	darkorange (255, 140, 0)	darkorchid (153, 50, 204)	darkred (128, 0, 0)	darksalmon (233, 150, 122)	darkseagreen (143, 188, 143)	darkslateblue (72, 61, 139)
darkslategray (47, 79, 79)	darkturquoise (0, 206, 208)	darkviolet (148, 0, 211)	deeppink (255, 20, 147)	deepskyblue (0, 191, 255)	dimgray (105, 105, 105)	dodgerblue (30, 144, 255)
firebrick (178, 34, 34)	floralwhite (255, 250, 240)	forestgreen (34, 139, 34)	fuchsia (255, 0, 255)	gainsboro (220, 220, 220)	ghostwhite (248, 248, 255)	gold (255, 215, 0)
goldenrod (218, 165, 32)	gray (128, 128, 128)	green (0, 128, 0)	greenyellow (173, 255, 47)	honeydew (240, 255, 240)	hotpink (255, 105, 180)	indianred (205, 92, 92)
indigo (75, 0, 133)	ivory (255, 255, 240)	khaki (240, 230, 140)	lavender (230, 230, 250)	lavenderblush (255, 240, 245)	lawngreen (124, 252, 0)	lemonchiffon (255, 250, 205)
lightblue (173, 216, 230)	lightcoral (240, 128, 128)	lightcyan (224, 255, 255)	lightgoldenrodyellow (250, 250, 210)	lightgreen (144, 238, 144)	lightgray (211, 211, 211)	lightpink (255, 182, 193)
lightsalmon (255, 160, 122)	lightseagreen (32, 178, 170)	lightskyblue (135, 206, 250)	lightslategray (119, 136, 153)	lightsteelblue (176, 196, 222)	lightyellow (255, 255, 224)	lime (0, 255, 0)
limegreen (50, 205, 50)	linen (250, 240, 230)	magenta (255, 0, 255)	maroon (128, 0, 0)	mediumaquamarine (102, 205, 170)	mediumblue (0, 0, 205)	mediumorchid (186, 85, 211)
mediumpurple (147, 112, 216)	mediumseagreen (60, 179, 113)	mediumslateblue (123, 104, 238)	mediumspringgreen (0, 250, 154)	mediumturquoise (72, 209, 204)	mediumvioletred (199, 21, 133)	midnightblue (25, 25, 112)
mintcream (245, 255, 250)	mistyrose (255, 228, 225)	moccasin (255, 228, 181)	navajowhite (255, 222, 173)	navy (0, 0, 128)	oldlace (253, 245, 230)	olive (128, 128, 0)
olivedrab (104, 142, 35)	orange (255, 165, 0)	orangered (255, 69, 0)	orchid (218, 112, 214)	palegoldenrod (238, 232, 170)	palegreen (152, 251, 152)	paleturquoise (175, 238, 238)
palevioletred (216, 112, 147)	papayawhip (255, 239, 213)	peachpuff (255, 218, 185)	peru (205, 133, 63)	pink (255, 192, 203)	plum (221, 160, 221)	powderblue (176, 224, 230)
purple (128, 0, 128)	red (255, 0, 0)	rosybrown (188, 143, 143)	royalblue (65, 105, 225)	saddlebrown (139, 69, 19)	salmon (250, 128, 114)	sandybrown (244, 164, 96)
seagreen (46, 139, 87)	seashell (255, 245, 238)	sienna (160, 82, 45)	silver (192, 192, 192)	skyblue (135, 206, 235)	slateblue (106, 90, 205)	slategray (112, 128, 144)
snow (255, 250, 250)	springgreen (0, 255, 127)	steelblue (70, 130, 180)	tan (210, 180, 140)	teal (0, 128, 128)	thistle (216, 191, 216)	tomato (255, 99, 71)
turquoise (64, 224, 208)	violet (238, 130, 238)	wheat (245, 222, 179)	white (255, 255, 255)	whitesmoke (245, 245, 245)	yellow (255, 255, 0)	yellowgreen (154, 205, 50)

Tabelle 1: Farben als RGB-Tripel.

3.2 Literale in Java (Vertiefung)



Literale (Konstanten) in Java:

Die Literale für Ganzzahlen lassen sich in vier unterschiedlichen **Stellenwertsystemen** direkt angeben. Das natürlichste ist das Dezimalsystem (auch Zehnersystem genannt), bei dem die Literale aus den Ziffern "0" bis "9" bestehen. Zusätzlich existieren die Binär-, Oktal- und Hexadezimalsysteme, welche den Zahlen zu den Basen 2, 8 und 16 entsprechen. Bis auf Dezimalzahlen beginnen die Zahlen in anderen Formaten mit einem besonderen Präfix:

Präfix	Stellenwertsystem	Basis	Darstellung von 1
0b oder 0B	binär	2	0b1 oder 0B1
0	oktal	8	01
kein	dezimal	10	1
0x oder 0X	hexadezimal	16	0x1 oder 0X1

Ein hexadezimaler Wert beginnt mit 0x oder 0X. Da zehn Ziffern für 16 hexadezimale Zahlen nicht ausreichen, besteht eine Zahl zur Basis 16 zusätzlich aus den Buchstaben a bis f (bzw. A bis F).

Ein oktaler Wert beginnt mit dem Präfix 0. Mit der Basis 8 werden nur die Ziffern 0 bis 7 für oktale Werte benötigt.

Für Dualzahlen (also Binärzahlen zur Basis 2) wurde eine neue Notation in **Java 7** eingeführt. Das Präfix ist 0b oder 0B. Es sind in diesem Fall nur die Ziffern 0 und 1 erlaubt.

Beispiel 1: Der folgende Programmausschnitt gibt **Dezimal-, Binär-, Oktal- und Hexadezimalzahlen** aus:

```
System.out.println( 1243 );           // 1243
System.out.println( 0b10111011 );     // 187

System.out.println( 01230 );           // 664
System.out.println( 0xcafebabe );     // -889275714

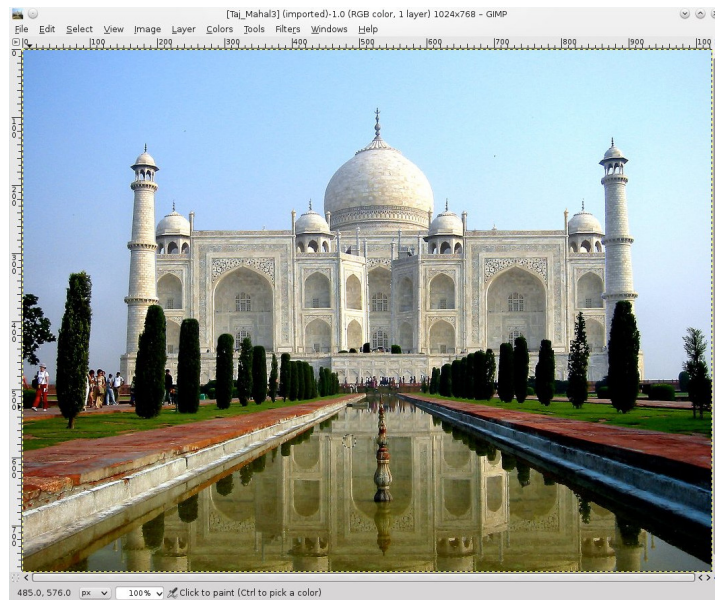
System.out.println( 0xC0B0L );         // 49328
System.out.println( 0xFF );           // 255
```

In Java-Programmen sollten Oktalzahlen mit Bedacht eingesetzt werden. Wer aus optischen Gründen eine Zahl mit der 0 (Nullen) linksbündig auffüllt, erlebt eine Überraschung:

```
int i = 118; int j = 012; // Oktal 012 ist dezimal 10
```

3.3 Abmessungen der 2D-Bilder

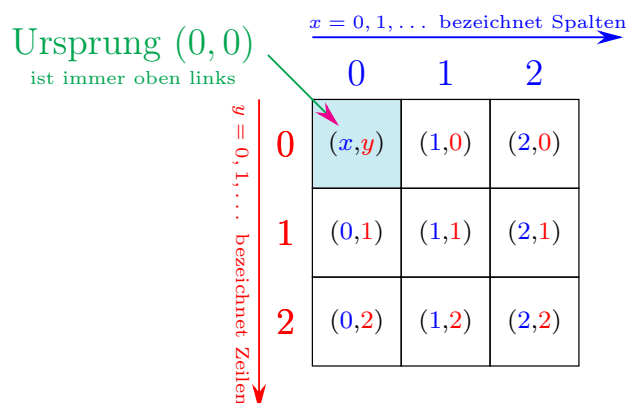
2D-Bilder haben eine Breite und eine Höhe, die im allgemeinen in Pixeln angegeben sind. Im folgenden Bild sind sämtliche Details wie Typ (oder Format: png, jpg ...), Grösse (in Bits, Bytes, etc.) sowie Breite und Höhe (in Pixel) der Bilddatei ersichtlich.



Abmessungen einer Bilddatei wird in Breite \times Höhe Format allgemein dargestellt. Im obigen Bild sehen wir die Abmessungen 1024×768 , was bedeutet, dass die Breite des Bildes 1024 Pixel und dessen Höhe 768 Pixel ist.

3.4 Darstellung von 2D-Bildpixeln

Für ein 2D-Bild werden Pixel in Zeilen und Spalten angeordnet. Der Ursprung (Ausgangspixel) des Bildes ist **immer** an der Koordinate (0,0). Die Abbildung unten hat die Masse von 3×3 Pixel.



So können wir ein Pixel (x,y) als $P_{x,y}(A,R,G,B)$ bezeichnen, wobei (x,y) die Koordinate des Pixels ist und A, R, G, B die Alpha-, Rot-, Grün- und Blau-Werte des Pixels bezeichnen.

Beispiel 2: $P_{0,0}(204,255,20,147)$ stellt ein Pixel bei Koordinaten $(0,0)$ mit $A = 204, R = 255, G = 20$ und $B = 147$.

3.5 Voraussetzungen



Beachten Sie !

Damit man das unten stehende Programm ausführen könnte, muss im Ordner Images des Projektverzeichnisses eine Bilddatei mit dem Namen `Sample_204_255_20_147.png` vorhanden sein !

Im Folgenden wird davon ausgegangen, dass Sie das vorherige Projekt 2 abgeschlossen haben. In diesem Projekt wurde demonstriert, wie man mit Java ein Bild aus einer Bilddatei einliest und es auch in eine Bilddatei schreibt. Der Java-Code, den wir in diesem Projekt verwenden werden:

```
1 import java.io. File ;
2 import java.io. IOException;
3 import java.awt.image. BufferedImage;
4 import javax.imageio. ImageIO;
5
6 public class GetSetPixels {
7     public static void main(String args[]) throws IOException {
8         BufferedImage img = null;
9         File f = null;
10
11         //read image
12         try {
13             f = new File("Images/Sample_204_255_20_147.png");
14             img = ImageIO.read(f); }
15         catch (IOException e) {
16             System.out.println(e); }
17
18         // ein Code kommt her ...
19
20     } // end main
21 } // end class GetSetPixels
```

Java-Programm zum Einlesen aus der Bilddatei "Images/Sample_204_255_20_147.png"

Nach den gewöhnlichen import-Befehlen wird in der `main()`-Methode des Quellencodes der Pfad zur Bilddatei `Images/Sample_204_255_20_147.png` definiert und dieser dem `File`-Objekt `f` zugewiesen:

```
f = new File("Images/Sample_204_255_20_147.png");
```

Danach wird die Bilddatei "Sample_204_255_20_147.png" eingelesen und in der Variablen `img` gespeichert:

```
img = ImageIO.read(f);
```

3.6 Wie ermittelt man Bilddimensionen (Abmessungen)?

Um die Masse der Bilddatei vor deren Einlesen (!) zu ermitteln, haben wir beim letzten Projekt 2.6 2.7 die Klasse `MyImageInfo.java` (s. 2, 3) benutzt. Alternativ können wir zwei Methoden `getWidth()` und/oder `getHeight()` verwenden, erst aber, nachdem die Bilddatei bereits eingelesen worden ist. Dazu erstellen wir zwei Integer-Variablen `Breite` und `Höhe`:

```
int width = img.getWidth();
int height = img.getHeight();
```

3.7 Aufgabe 2 zum Kapitel 3

Bauen Sie die zwei oben stehenden Methoden `getWidth()`, `getHeight()` ins Programms 2 ein und ermitteln Sie somit die Dimensionen der Bilddatei "Images/Sample_204_255_20_147.png". Prüfen Sie das Resultat sowohl mittels 3 als auch z.B. mit Hilfe des GIMP-Grafikprogramms nach.

3.8 Wie ermittelt man den Pixelwert?

Um den Pixelwert zu ermitteln, werden wir die `getRGB(x, y)`-Funktion verwenden. Dieser Methode übernimmt als Parameter die Koordinaten des Pixels und liefert einen ganzzahligen Wert, der positiv und auch negativ sein kann. Um den Pixelwert zu speichern, erzeugen wir eine Integer-Variable:

```
int p = img.getRGB(x,y);
```

Da z.B. die `Sample_204_255_20_147.png`-Datei ein einzelnes Pixel-Bild ist (s. oben), müssen wir `(x, y)` durch `(0,0)` ersetzen, um dieses Pixel mit Hilfe von `getRGB(x, y)` zu lesen.

So wird der obige Code geschrieben werden als:

```
int p = img.getRGB(0,0);
```

3.9 Wie ermittelt man Alpha-, Red-, Green- und Blau-Werte des Pixels?

Nachdem der Pixelwert ermittelt und in der Integer-Variablen `p` (s. oben) gespeichert worden ist, können wir individuelle Werte für jeden der vier Pixel-Komponenten (d.h. ARGB-Werte) erhalten. Da jeder ARGB-Wert ganzzahlig im Bereich von 0 bis 255 ist, werden wir dafür 4 Integer-Variablen `a`, `r`, `g` und `b` erstellen.

Wir haben bereits gesehen, dass die ALPHA-Werte 8 Bits vom Index 24 bis zum 31 besetzen. Um die ALPHA-Bits zu erhalten, müssen wir daher zuerst die 32 Bits der Pixel um 24-Positionen nach rechts verschieben und dann bitweise mit `0xFF` multiplizieren (d.h. mittels UND-Operator "&" verknüpfen). Wenn wir diesen Rechenschaltvorgang durchführen, bringen wir die Bits auf die rechte 8-Bit-Position. Anschliessendes bitweises UND-Verknüpfung mit `0xFF` ergibt dann den Wert der Alpha-Komponente, wofür wir hier interessiert sind. Um den gewünschten Alpha-Wert zu erhalten, **schreiben wir** (s. auch das entsprechende Kapitel im Script!):

```
int a = (p >> 24) & 0xFF;
```

Hinweis! `0xFF` ist die hexadezimale Darstellung des Dezimalwerts $255_{10} = 1111\ 1111_2$.

In ähnlicher Weise belegen wir die 8 ROTEN-Bits, vom Index 16 zum Index 23. Um die roten Bits zu erhalten, müssen wir zuerst die 32 Bits der Pixel um die 16 Position verschieben und sie danach bitweise mit `0xFF` multiplizieren. Dazu schreiben wir:

```
int r = (p >> 16) & 0xFF;
```

Ebenso belegen die GRÜN-Bits 8 Bits vom Index 8 zum Index 15. Um die GRÜNEN Bits zu erhalten, müssen wir die 32 Bits der Pixel nach rechts verschieben und dann mit bitweisem UND mit `0xFF` verknüpfen.

```
int g = (p >> 8) & 0xFF;
```

Und um die BLAU-Bits zu erhalten, die 8 Bits von Index 0 bis zum Index 7 einnehmen, **müssen die Pixelbits nicht nach rechts verschoben werden**. Dies liegt daran, dass die BLAU-Bits bereits die rechten 8 Bits belegen. Wir werden also einfach den Pixelwert durch bitweises UND mit `0xFF` verknüpfen.

```
int b = p & 0xFF;
```

Nun sieht unser Code wie folgt aus:

```
1 import java.io. File ;
2 import java.io. IOException;
3 import java.awt.image. BufferedImage;
4 import javax.imageio. ImageIO;
```



```

5 public class GetSetPixels {
6     public static void main(String args[]) throws IOException {
7         BufferedImage img = null;
8         File f = null;
9
10        // read image
11        try {
12            f = new File("Images/Sample_204_255_20_147.png");
13            img = ImageIO.read(f);
14        } catch(IOException e) {
15            System.out.println(e);
16        }
17
18        // get image width and height
19        int width = img.getWidth();
20        int height = img.getHeight();
21
22        // get Pixelwert
23        int p = img.getRGB(0,0);
24
25        // get Alpha-Wert
26        int a = (p >> 24) & 0xFF;
27
28        // get Red-Wert
29        int r = (p >> 16) & 0xFF;
30
31        // get Green-Wert
32        int g = (p >> 8) & 0xFF;
33
34        // get Blue-Wert
35        int b = p & 0xFF;
36
37        // ein Code kommt her ...
38
39    } // main() ends here
40 } // class GetSetPixels ends here

```

Java-Programm zur Bestimmung der ARGB-Werte eines Pixels

3.10 Aufgabe 3 zum Kapitel 3

Verwenden Sie den obigen Programm Quellcode und lassen Sie somit die ARGB-Werte des Pixels auf die Konsole ausgeben. Prüfen Sie das Ergebnis mit Hilfe eines grafischen Standardprogramms nach. Dabei muss der Vergrößerungsfaktor auf ca. 2000% eingestellt werden, damit Sie das Bild auf dem Computermonitor sehen und seine Farbe optisch identifizieren könnten. Vergleichen Sie die von Ihnen bestimmten RGB-Pixelwerte mit der obigen RGB-Tabelle 1 und stellen Sie somit den Namen der Pixelfarbe fest! Benutzen Sie dabei auch das online-Programm <http://www.tydac.ch/color/>. Typische Programmausgabe sollte beispielsweise wie folgt aussehen:

```

p                = 110011001111111100010100100100112 = -85569828510
p >> 24          = 11111111111111111111111111110011002 = -5210
0xFF             = 000000000000000000000000011111112
a = (p >> 24) & 0xFF = 110011002 = 20410
p >> 16          = 111111111111111111110011001111112 = -1305710
0xFF             = 000000000000000000000000011111112
r = (p >> 16) & 0xFF = 11111112 = 25510
p >> 8           = 11111111110011001111111111000101002 = -334257210

```

[illegible]

3.11 Wie setzt man den Pixelwert?

Um das Projekt einfach zu halten, können Sie den Wert von Alpha, Rot, Grün und Blau auf 255, 100, 150 und 200 festlegen. Dafür schreiben wir:

a = 255; r = 100; g = 150; b = 200;

Um die individuellen Pixelfarben mit den neuen ARGB-Werten zu belegen, setzen wir zunächst `p` auf 0:

$p = 0;$

Hinweis! p ist eine Integer-Variable, die den Wert des Pixels hält.

Wir wissen, dass 8 Bits der ALPHA-Komponente die Bitpositionen vom Index 24 bis zum Index 31 einnehmen. Wir werden also den Alpha-Wert um 24 Stellen positionieren und bitweise mittels ODER-Operator (d.h. "|") mit dem p-Wert verknüpfen:

```
p = p | (a << 24);
```

In ähnlicher Weise belegen 8 Bits der ROT-Komponente die Bitposition von Index 16 bis zum Index 23. So werden wir den roten Wert um 16 Positionen verschieben und bitweise mittels ODER-Operators mit dem p-Wert verknüpfen:

```
p = p | (r << 16);
```

Gleichermassen belegen 8 Bits der GRÜN-Komponente die Bitpositionen vom Index 8 bis zum Index 15. So werden wir den grünen Wert um 8 Positionen verschieben und bitweise mittels ODER-Operators mit dem p-Wert verknüpfen:

```
p = p | (g << 8);
```

Die BLUE-Komponente hingegen muss **nicht nach links verschoben werden**. Wir werden sie einfach bitweise durch ODER mit p verknüpfen:

$p = p \mid b;$

Natürlich kann man die obigen drei Codezeilen ganz kompakt in einer einzigen Zeile schreiben:

p = (a << 24) | (r << 16) | (g << 8) | b;

Um diesen Pixelwert p bei den Koordinaten (x, y) zu setzen, verwenden wir die Funktion

```
setRGB(x, y, p)
```

Da "Images/Sample_204_255_20_147.png" Bild ein einzelnes Pixelbild ist, werden wir hier (x, y) durch (0,0) ersetzen. Dazu schreiben wir in diesem Fall:

```
img.setRGB(0, 0, p);
```

3.12 Aufgabe 4 zum Kapitel 3:

Erstellen Sie ein ausführbares Programm, welches: (i) liest die Bilddatei "Images/Sample_204_255_20_147.png" ein, (ii) ermittelt seine Abmessungen (d.h. Höhe und Breite) in Pixel, (iii) bestimmt individuelle ARGB-Werte des Pixels, (iv) setzt sie auf die neuen Werte A = 255, R = 100, G = 150 und B = 200 und danach (v) schreibt das Bild in eine neue Bilddatei "Images/Sample_255_100_150_200.png".

Hinweis! Verwenden Sie dazu unter anderem die oben aufgeführten Methoden `getRGB(x,y)` und `setRGB(x,y,p)`. Überprüfen Sie die neuen Eigenschaften des Bildes mit Hilfe eines grafischen Standardprogramms.

4 Konvertierung eines Farbenbildes in ein Graustufenbild

In diesem Projektteil werden wir lernen, ein Farbenbild in ein Graustufenbild mit Java-Programmiersprache zu konvertieren.

4.1 Voraussetzungen

Es wird davon ausgegangen, dass Sie die vorherigen Projekte 2 und 3 abgeschlossen haben.

4.2 Farbaufnahme zum Graustufenbild

Das Konvertieren eines Farbenbildes in Graustufenbild ist sehr einfach. Alles, was wir tun müssen, ist, 3 folgende einfache Schritte für jedes Pixel des Bildes zu wiederholen:

1. Holen Sie sich den RGB-Wert des Pixels.
2. Finden Sie den Durchschnitt von RGB, d.h. $Avg = \frac{1}{3}(R + G + B)$
3. Ersetzen Sie den R-, G- und B-Wert des Pixels mit dem in Schritt 2 berechneten Mittelwert (Avg).

Beispiel 1: Betrachten Sie ein Farbenpixel mit den folgenden Werten: $A = 255$, $R = 100$, $G = 150$, $B = 200$, wobei A, R, G und B den Alpha-, Rot-, Grün- und Blau-Wert des Pixels darstellen.

Merken! Jeder der vier ARGB-Parameter hat einen ganzzahligen Wert im Bereich von 0 bis 255.

Um das Farbenpixel in Graustufen-Pixel umzuwandeln, müssen wir zuerst den Mittelwert von R, G und B finden. Der gesuchte Durchschnitt ist $Avg = \frac{1}{3}(R + G + B) = \frac{1}{3}(100 + 150 + 200) = 150$. Nun werden wir die Werte von R, G und B jeweils durch den berechneten Mittelwert Avg ersetzen, so dass nun die neuen Pixelwerte betragen: $A = 255$, $R = 150$, $G = 150$, $B = 150$.

Hinweis 1: Wir müssen den Alpha-Wert nicht ändern, da er nur die Transparenz des Pixels steuert.

Hinweis 2: Im Allgemeinen sind wir natürlich nur an einem Integer-Wert von Avg interessiert. Daher muss der berechnete Mittelwert normalerweise in eine Integerzahl gecastet (umgewandelt) und in einer Variablen Avg vom Datentyp int gespeichert werden.

4.3 Aufgabe zum Kapitel 4

Schreiben Sie eine Klasse GraustufenImage zur Konvertierung der Bilddatei "Images/Taj_Mahal4.png" in ein Graustufenbild, das in eine neue Bilddatei "Images/Taj_Mahal4_grau.png" zu schreiben ist. Als Leistungsnachweis legen Sie Ihrer Lösung der Programmquellcode sowie die Bilddatei "Images/Taj_Mahal4_grau.png" bei.

Tip: Wir wissen, dass jedes Bild aus (Breite \times Höhe) Pixeln besteht, die wir auch gleichzeitig anhand ihrer 2D-Koordinaten auffassen können. Man ermittelt die Abmessungen width und height des Bildes und erstellt dann zwei Variablen x,y ($x = 0, \dots, width$ und $y = 0, \dots, height$). Danach verwendet man zwei Schleifen, um jedes Pixel des Pixel-Gitter zu durchlaufen. Dazu schreiben wir:

```
for (int y = 0; y < height; y++) {  
    for (int x = 0; x < width; x++) {  
        // ein Code kommt her ...  
    }  
}
```

Innerhalb der innersten for-Schleife erhält man zunächst den Pixelwertwert p an den Koordinaten (x, y) durch den Aufruf: $p = \text{getRGB}(x, y)$. Mit Hilfe vom Pixelwert p lassen sich dann die Alpha-, Rot-, Grün- und Blau-Werte extrahieren. Zu diesem Zweck erstellt man 5 Integer-Variablen p, a, r, g und b, um die ARGB-Wert zu halten. Anschliessend muss der Mittelwert (Avg) für jedes Pixel berechnet und mit Avg die RGB-Pixelwerte belegt werden. Dazu benutzt man die beim Teilprojekt 3 aufgeführten Methoden $\text{getRGB}(x, y, p)$ und $\text{setRGB}(x, y, p)$. Anschliessend schreibt man das so abgeänderte Image in die neue Bilddatei, deren Eigenschaften z.B. mit Hilfe des GIMP-Grafikprogramms zu überprüfen sei.

5 Konvertierung eines Farbenbildes in ein Negativenbild

In diesem Projekt werden wir lernen, ein Farbenbild in sein Negativ mit Java-Programmiersprache zu konvertieren.

5.1 Voraussetzungen

Es wird davon ausgegangen, dass Sie die vorherigen Projekte 2 bis 4 abgeschlossen haben.

5.2 Farbenbild zum Negativenbild

Das Umwandeln eines Farbenbildes in das Negativ ist sehr einfach. Alles, was wir tun müssen, ist, 3 einfache Schritte für jedes Pixel des Bildes zu wiederholen:

1. Holen Sie sich den RGB-Wert des Pixels.
2. Berechnen Sie die neuen RGB-Werte wie folgt:

$$R = 255 - R, \quad G = 255 - G, \quad B = 255 - B$$

3. Speichern Sie die neu berechneten RGB-Werte wieder im Pixel.

Beispiel 1: Betrachten Sie ein Farbenpixel mit den folgenden Werten: $A = 255$, $R = 100$, $G = 150$, $B = 200$, wobei A , R , G und B den Alpha-, Rot-, Grün- und Blau-Wert des Pixels darstellen.

Merken! Jeder der vier ARGB-Parameter hat einen ganzzahligen Wert im Bereich von 0 bis 255.

Um das Farbenpixel in negativ umzuwandeln, werden wir die Werte von R , G und B von 255 subtrahieren:

$$R = 255 - 100 = 155, \quad G = 255 - 150 = 105, \quad B = 255 - 200 = 55$$

Auf dieser Weise werden die neuen ARGB-Werte sein: $A = 255$, $R = 155$, $G = 105$, $B = 55$.

Hinweis! Wir müssen den Alpha-Wert nicht ändern, da er nur die Transparenz des Pixels steuert.

5.3 Aufgabe zum Kapitel 5

Diese Aufgabe ähnelt der vorherigen Aufgabe 4.3 beim Projekt 4. Schreiben Sie eine Klasse `NegativImage` zur Konvertierung der Bilddatei "Images/Taj_Mahal4.png" in ein Negativenbild, das in eine neue Bilddatei "Images/Taj_Mahal4_negativ.png" zu schreiben ist. Als Leistungsnachweis legen Sie Ihrer Lösung der Programmquellcode sowie die Bilddatei "Images/Taj_Mahal4_negativ.png" bei.

6 So erstellen Sie ein zufälliges Pixelbild in Java

In diesem Projektteil werden wir lernen, wie man ein zufälliges Pixelbild mit Java-Programmiersprache erstellt.

6.1 Voraussetzungen

Es wird davon ausgegangen, dass Sie die vorherigen Projektteile 2 bis 5 abgeschlossen haben.

6.2 Farbenbild auf zufällige Pixelbild

Das Erstellen eines Zufallsbildes basiert auf Zufallszahlen. Wir müssen die folgenden 3 Schritte durchführen, um das zufällige Pixelbild zu erhalten.

1. Erstellen Sie eine neue Klasse mit dem Namen `RandomImage.java`
2. Öffnen Sie die Klassendatei und importieren Sie in erster Linie die erforderlichen Klassen:

```
import java.io. File ;
import java.io. IOException;
import java.awt.image. BufferedImage;
import javax.imageio. ImageIO;
```

3. Legen Sie die Grösse des Bildes fest. Für dieses Projekt werden wir Breite = 640 (= width) und Höhe = 320 (= height) nehmen.
4. Erstellen Sie ein `BufferedImage`-Objekt `img` sowie ein `File`-Objekt `fout`, das den Pfad zur Bilddatei "Images/Random_Image.png" definiert:

```
import java.io. File ;
import java.io. IOException;
import java.awt.image. BufferedImage;
import javax.imageio. ImageIO;

public class RandomImage {
    public static void main(String args[]) throws IOException {

        int width = 640;
        int height = 320;
        BufferedImage img = new BufferedImage (width, height, BufferedImage.TYPE_INT_ARGB);
        File fout = new File ("Images/Random_Image.png");

        // Ein Code kommt her ...

    } // end main()
} // end class RandomImage
```

5. Generieren Sie zufällige Integer-Werte für Alpha-, Rot-, Grün- und Blau-Komponente und erzeugen Sie zufällige Pixel.

6.3 Random-Pixel-Code

Wir wissen, dass ein Bild aus Pixeln besteht, die wir in 2D koordinieren können. So erstellen wir zwei Variablen `x` und `y` und verwenden zwei für Schleifen, um jedes Pixel zu durchlaufen. Dazu schreiben wir (vgl. mit 4.3):

```
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        // ein Code kommt her ...
    }
}
```

6.4 Generierung von zufälligen Pixeln

Um eine Zufallszahl in Java zu generieren, verwenden wir die `random()`-Methode der `Math`-Klasse. Dies erzeugt einen Wert grösser oder gleich 0 und kleiner als 1. Das heisst, wir erhalten Werte wie 0.0 oder 0.1 oder 0.999 99 usw. Dabei werden wir niemals 1 bekommen.

Wir wissen auch, dass Alpha, Rot, Grün und Blau einen ganzzahligen Wert von 0 bis 255 annehmen können. Um einen Wert im Bereich von 0 bis 255 zu erhalten, multiplizieren wir zuerst die Zufallszahl mit 256 und wandeln das Ergebnis in ein Integer um. Um die zufällige Zahl zu erhalten, schreiben wir daher:

```
Math.random () * 256
```

die uns einen Wert grösser oder gleich 0 und kleiner als 256 gibt. Etwas wie 0 oder 123.45 oder 255.999 999. Es wird uns niemals 256 ergeben.

Jetzt den Integer-Wert zu erhalten wir den Float-Wert eingeben, wird ihren Datentyp `int`. Dazu schreiben wir:

```
(int) Math.random() * 256
```

Um zufällige und voneinander unabhängige Alpha-, Rot-, Grün- und Blau-Werte zu erzeugen, schreiben wir:

```
int a = (int) Math.random()*256; //alpha  
int r = (int) Math.random()*256; //red  
int g = (int) Math.random()*256; //green  
int b = (int) Math.random()*256; //blue
```

Beachte! Im obigen Code wird Alpha-Komponente auch mit dem zufälligen Wert $0 \leq a \leq 255$ belegt.

Nun setzen wir den neuen Pixelwert `p` und schreiben dazu:

```
p = (a << 24) | (r << 16) | (g << 8) | b;  
img.setRGB(x, y, p);
```

6.5 Bild schreiben

Nachdem die beiden `for`-Schleifen abgearbeitet haben, sollte das Bild (Image `img`) parat zum Schreiben in die neue Bilddatei stehen. Das Schreiben lässt sich mit dem folgenden wohl bekannten Code realisieren:

```
try {  
    ImageIO.write(img, "png", fout);  
} catch (IOException e) {  
    System.out.println(e);  
}
```

6.6 Aufgabe zum Kapitel 6

Schreiben Sie eine Klasse `RandomImage` zur Erzeugung einer Reihe der Bilddateien `"Images/Random_Image_1.png"`, `"Images/Random_Image_2.png"`, ..., `"Images/Random_Image_6.png"`. Für die 1., 2., 3., 4. und 5. Dateien belegen Sie die Alpha-Komponente mit den folgenden fixen `a`-Werten: 255, 200, 150, 50 bzw. 0. Für die 6. Datei `"Images/Random_Image_6.png"` hingegen setzen diesen (s. oben!) auf `a = (int) Math.random()*256`.

Vergleichen Sie mit Hilfe eines grafischen Windowsprogramms die Eigenschaften dieser sechs Bilder optisch miteinander. Als Leistungsnachweis legen Sie Ihrer Lösung der Programmquellcode sowie die folgenden Bilddateien bei: `"Images/Random_Image_1.png"`, ..., `"Images/Random_Image_6.png"`.

7 So erstellen Sie ein Spiegelbild in Java

In diesem Projekt lernen wir, ein Spiegelbild (s. die Abb. 3 und 4) mit Hilfe von Java zu erstellen.



Abb. 3: Das Originalbild.



Abb. 4: Das gespiegelte Bild (vgl. mit Abb. 3 links).

7.1 Voraussetzungen

Es wird davon ausgegangen, dass Sie die vorherigen Projektteile 2 bis 6 abgeschlossen haben.



Beachten Sie !

Damit man das unten stehende Programm ausführen könnte, muss im Ordner Images des Projektverzeichnisses eine Bilddatei mit dem Namen `Doctor_Strange.png` vorhanden sein !

Führen Sie nun die folgenden Vorbereitungsschritte 1 bis 2 aus.

1. Legen Sie vom BlueJ aus eine neue Klasse mit dem Namen `MirrorImage` an.
2. Machen Sie nachher die Datei `MirrorImage.java` im BlueJ-Editor auf und nehmen Sie darin die folgenden Änderungen vor.

7.2 Spiegelbild erzeugen

Um ein Spiegelbild zu erzeugen, müssen Sie das Bild pixelweise wie folgt aufbauen:

1. Importieren Sie in erster Linie alle erforderlichen Klassen.
2. Lesen Sie das Quellbild `Doctor_Strange.png` ins zuvor deklarierte `BufferedImage`-Objekt `simg`.
3. Ermitteln Sie die Dimensionen `width` und `height` des Quellbildes `Doctor_Strange.png`.
4. Erstellen Sie für das Spiegelbild ein neues Objekt `mimg` vom Datentyp `BufferedImage`. Legen Sie dabei die Breite von `mimg` **auf die doppelte Bildbreite (`width`) und die gleiche Höhe (`height`)** wie bei `simg`:

```
BufferedImage mimg = null;  
// Ein Code kommt her ...  
BufferedImage mimg = new BufferedImage(width*2, height, BufferedImage.TYPE_INT_ARGB);
```

5. Kopieren Sie das Quellbild pixelweise auf das Zielbild `simg` (s. Kapitel 7.3 unten) und speichern Sie es anschliessend in die Bilddatei `Doctor_Strange_spiegel.png`.

7.3 Spiegelbildcode

Wir wissen, dass ein Bild aus Pixeln besteht, die wir in 2D-Koordinatensystem auffassen können. So werden wir drei Schleifenvariablen, `lx`, `rx` und `y` erstellen, um die einzelnen Pixel zu durchlaufen. Dazu schreiben wir:

```
for (int y = 0; y < height; y++) {  
    for (int lx = 0, rx = width*2 - 1; lx < width; lx++, rx--) {  
        // lx/rx starts from the left/right side of the image  
        // Ein Code kommt her ...  
    }  
}
```

Nachdem der Quellpixelwert bei `simg` an einer beliebigen Koordinate (`lx`, `y`) erhalten worden ist, setzen wir ihn in der Spiegeldarstellung bei `mimg` gleichzeitig an zwei Stellen mit Koordinaten (`lx`, `y`) und (`rx`, `y`), was schlussendlich den erwünschten Spiegeleffekt ergibt:

```
int p = simg.getRGB(lx, y); // Pixelwert an der Koordinate (lx, y) abfragen und speichern  
mimg.setRGB(lx, y, p); // Spiegel-Pixel-Wert p an der Koordinate (lx, y) setzen  
mimg.setRGB(rx, y, p); // Spiegel-Pixel-Wert p an der Koordinate (rx, y) setzen
```

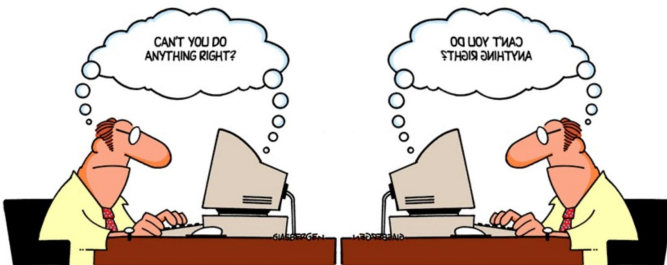


Abb. 5: Das ganze (doppelt) gespiegelte Bild.



Abb. 6: Die "Hälfte" des gespiegelten Bildes.

Nachdem die beiden obigen `for`-Schleifen abgearbeitet haben, steht das Bild `mimg` (s. Abb. 5 oben) schon parat zum Schreiben in die neue (Spiegel)Bilddatei. Das Schreiben an sich lässt sich dabei mit dem Standardcode aus dem Projektteil 2 realisieren. Hierher speichern wir jedoch **nicht das ganze Image** `mimg`, sondern nur seine rechte "Hälfte" (s. Abb. 6 oben). Die letztere entspricht einem rechtwinkligen Imagebereich des `mimg` von der Breite `width` und der Höhe `height`, beginnend mit der Koordinate (`width`, 0) (oben links) **als neuer Bild-Ursprung**. Das Ausschneiden (beachte die Argumente des `getSubimage(...)`-Befehls!) und Speichern ins Output-File `fout` kann mit dem folgenden Java-Befehl ganz kompakt implementiert werden:

```
ImageIO.write(mimg.getSubimage(width, 0, width, height), outFormat, fout);
```

7.4 Aufgaben zum Kapitel 7

1. Schreiben Sie eine Klasse `MirrorImage` zur Erzeugung eines gespiegelten Bildes.
2. Führen Sie das Programm für die Bilddatei "Images/Doctor_Strange.png" aus. Erzeugen Sie somit die Bilddatei "Images/Doctor_Strange_spiegel.png" und überprüfen Sie deren Eigenschaften mit Hilfe des grafischen Windowsprogramms GIMP.
3. Lassen Sie sich im Internet darüber informieren oder finden Sie es selbst heraus, wie die oben aufgeführte (und bisher von uns nicht verwendete!) verallgemeinerte Form der folgenden `for`-Schleife mit **zwei (!) statt einer Schleifenvariablen** `lx` und `rx` funktioniert:

```
for (int lx = 0, rx = width*2 - 1; lx < width; lx++, rx--) {  
    // Ein Code kommt her ...  
}
```

4. Untersuchen Sie die Programmstruktur der von Ihnen geschriebenen Klasse `MirrorImage` und erstellen Sie dafür mit Hilfe des draw-Programms, welches Ihnen im Internet unter www.draw.io frei zur Verfügung steht, das entsprechende PAP-Diagramm. Exportieren Sie dieses PAP-Diagramm direkt aus dem draw-Programm in eine pdf-Datei und speichern sie ins Unterverzeichnis `Images` des Projektverzeichnis unter dem Namen `"PAP_MirrorImage.pdf"`. Legen Sie Ihrer Lösung den Ausdruck dieses Files bei.
5. Experimentieren Sie mit verschiedenen png/jpg-Bilddateien sowie Ausschnittsbereichen Ihrer Wahl. Achten Sie jeweils darauf, welcher Bilddateityp vorliegt und ob der Alphakanal im Bild eingebettet ist.
6. Untersuchen Sie jeweils die Eigenschaften der Spiegelbilder mit Hilfe des GIMP-Programms.
7. **Vertiefung:** Ändern Sie Ihre Klasse `MirrorImage` so ab, um die Filenamen des Original- sowie des gespiegelten Bildes über ein grafisches Menü, das zuvor bei der Klasse (2) verwendet wurde, wählen zu können. Überlegen Sie sich ausserdem, **nach welchem Algorithmus** die Spiegelung **nicht** an der Y-, sondern an der X-Achse des Bildes (oder sogar gleichzeitig an dessen X- **und** an Y-Achsen) realisiert werden kann.