Funktionale und objektorientierte Programmierkonzepte

Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.2

Übungsblatt 6

Themen:	Klassen, Interfaces, Schleifen, Arrays, Methoden
Relevante Folien:	Objektorientierte Abstraktion, Methoden
Abgabe der Hausübung:	07.12.2018 bis 23:55 Uhr

Wichtig und verpflichtend! Dokumentation in Java

Sie erinnern sich sicher an die Dokumentation im Racket-Teil der Veranstaltung. Dort wurde der Typ, sowie eine Beschreibung der Rückgabe über jeder Funktionsdefinition als Kommentar angegeben.

```
;; Type: number number -> number
;; Returns: Euclidean norm of the vector (x,y)
(define (euclid2 x y)
(sqrt (+ (* x x) (* y y))))
```

Mittels Javadoc und den Tags `@param` und `@return` wollen wir nun auch im Java-Teil eine geeignete Dokumentation unserer Methoden vornehmen. Das Javadoc können Sie automatisch vor jeder Methode mittels `/**` gefolgt von der Enter-Taste generieren. Das Racket-Beispiel oben könnten wir dann folgendermaßen in Java übersetzen:

```
/**
 * @param x first component of two-dimensional vector  (x,y)
 * @param y second component of two-dimensional vector (x,y)
 * @return Euclidean norm of the vector (x,y)
*/
double euclid2(float x, float y){
   return Math.sqrt(x*x + y*y); }
```

Genauso wie in den Teilen mit Racket, ist die Dokumentation mit Vertrag Pflicht für alle Javamethoden! Orientieren Sie sich dabei an obigem Beispiel.

Auf Tests wie Sie sie in Racket mit `check-expect` und `check-within` geschrieben haben, können Sie im Java-Teil zunächst verzichten. Die Möglichkeit Ihre Methoden zu testen lernen wir später kennen und werden an geeigneter Stelle nochmals darauf hinweisen, wenn Tests von Ihnen erwartet werden.

V Vorbereitende Übungen

V1 Grundbegriffe ☆ ☆ ☆

Erklären Sie kurz in eigenen Worten die Unterschiede der folgenden Konzepte zueinander:

1. Klasse vs. Objekt
2. Objekt- vs. Klassenmethoden
3. Abstrakte Klassen vs. Interfaces
4. Überladen von Methoden vs. Überschreiben von Methoden

V2 Lambda I: Java → Racket ★ ☆ ☆

In der Vorlesung haben Sie das folgende Interface kennengelernt:

```
public interface IntToDoubleFunction{
  double applyAsDouble(int n); }
```

Sie haben ebenfalls ein Beispiel mittels Lambda-Ausdrücken in Java gesehen:

```
IntToDoubleFunction fct2 = x -> x * 10;
double z = fct2.applyAsDouble(11);
```

Setzen Sie dieses Beispiel nun mittels Lambda-Ausdrücken in Racket um. Die Methode `applyAsDouble` soll ebenfalls implementiert werden.

V3 Lambda II: Racket → Java ★ ☆ ☆

Betrachten Sie folgendes Beispiel in Racket:

```
(define (foo x) (lambda (c) (> (* x x) c)))
```

Setzen Sie dieses Beispiel mittels FunctionalInterface und Lambda-Ausdruck in Java um.

V4 Die Würfel sind gefallen! ★ ☆ ☆

Mit der Funktion `Math.random()` können Sie eine Zufallszahl im Bereich 0 (inklusive) und 1 (exklusive) erzeugen. Schreiben Sie nun eine Methode `void diceRoll()`.

Diese soll einen Würfelwurf simulieren und die gewürfelte Augenzahl auf der Konsole zurückgeben. Dabei soll der Würfel fair sein, dass heißt alle Augenzahlen sollen mit identischer Wahrscheinlichkeit auftreten.

Hinweise: Überlegen Sie sich, wie Sie die erzeugten Zahlen aus dem Intervall $[0, 1)$ auf die diskrete Menge $\{1, 2, 3, 4, 5, 6\}$ abbilden können. Mit der Funktion `Math.ceil()` können Sie zur nächstgrößeren, ganzen Zahl aufrunden.

V5 Schleifen ★ ☆ ☆

```
1 int x = 0;
2 while(Math.pow(x,2) <= 1000){
3   x++;  }
4 System.out.println(x);
```

(1) Welche Funktion erfüllt der oben stehende Code?

(2) Ersetzen Sie die `while`-Schleife einmal durch eine `for`- und einmal durch eine `do...while`-Schleife.

V6 Brumm, Brumm, Brumm ★ ☆ ☆

Schreiben Sie eine Klasse `Car` zur Repräsentation von Autos, die folgende Anforderungen erfüllen soll:

- Ein Auto hat einen Namen vom Typ *String* und einen Kilometerstand (*mileage*) vom Typ *double*. Beide Attribute sollen `private`, nicht `public`, sein.
- Der Konstruktor soll einen String als Parameter erhalten, der den Namen des Autos angibt. Der Konstruktor soll den Namen des Autos setzen und den Kilometerstand auf 0.0 setzen.
- Schreiben Sie die Methoden `public double getMileage()` und `public String getName()`. Diese liefern die entsprechenden Attribute der Klasse `Car` zurück.
- Schreiben Sie die Methode `public void drive(double distance)`, die eine Distanz in Kilometern als Argument erhält und auf den alten Kilometerstand addiert.

V7 Gleicher Abstand ★ ★ ☆

Schreiben Sie eine Methode `static boolean evenlySpaced(int a, int b, int c)`, welche genau dann `true` zurückliefert, wenn der Abstand zwischen dem kleinsten und dem mittleren Element genauso groß ist wie der Abstand zwischen dem mittleren und dem größten Element. Dabei kann jeder der Parameter `a`, `b` oder `c` das kleinste, mittlere oder größte Element sein. Die Klasse, zu der die Methode gehört, muss nicht implementiert werden.

V8 Aufsummieren ★ ★ ☆

Schreiben Sie eine Methode `void sumUp(int[] a)`, die ein Array `a` von Typ `int` erhält.

An Index $i \in$ `{0,...,a.length-i}` in `a` soll nun der neue Wert `a[0]+...+a[i]` geschrieben werden. Dabei bezeichnen `a[0],...,a[i]` die Werte in `a` unmittelbar vor dem Aufruf der Methode.

(Beispiel auf nächster Seite)

Übergeben wir der Funktion das folgende Array `a = [3, 4, 1, 9, -5, 4]`, so wird das Array folgendermaßen modifiziert:

→ `[3, 3+4, 3+4+1, 3+4+1+9, 3+4+1+9+(-5), 3+4+1+9+(-5)+4]`

→ `[3, 7, 8, 17, 12, 16]`

V9 Spieglein, Spieglein... ★★☆

Wir nennen eine Gruppe von Elementen in einem Array Spiegel, wenn sie irgendwo im Array nochmal auftaucht, nur in umgekehrter Reihenfolge. Beispielsweise ist im Array `[7,6,5,1,9,8,5,6,7]` ein Spiegel vorhanden und zwar `[7,6,5]`. Schreiben Sie eine Methode `int maxMirror(int[] arr)`. Diese bekommt ein Array übergeben und gibt die Länge des größten Spiegels im übergebenen Array zurück. Gibt es keinen Spiegel so wird einfach 0 zurückgeliefert.

Hinweis: Starten Sie mit zwei Zeigern auf dem ersten und dem letzten Element. Vergleichen Sie nun paarweise die Elemente und überlegen Sie sich, wann Sie die beiden Zeiger weiter in die Mitte bewegen.

V10 Matrix-Multiplikation Reloaded ★★☆

Erinnern Sie sich an die zweite Hausübung? Dort haben Sie die Matrixmultiplikation bereits in Racket implementiert. Wir wollen dies auch in Java umsetzen, und verwenden statt verschachtelten Listen nun zweidimensionale Arrays.

Der folgende Code stellt beispielsweise die Matrix $\begin{pmatrix} 5 & 8 \\ 1 & -3 \end{pmatrix}$ dar.

```
int[][] matrix = new int[2][2];
matrix[0][0] = 5;
matrix[0][1] = 8
matrix[1][0] = 1
matrix[1][1] = -3;

// alternativ und kuerzer: int[][] matrix = {{5,8},{1,-3}}
```

Sie sehen also, dass Sie einem Array in Java beliebig viele Dimensionen geben können.

Schreiben Sie eine Methode `int[][] matrixMul(int[][] mat1, int[][] mat2)`.

Die Methode bekommt zwei Matrizen, dargestellt durch zwei zwei-dimensionale Arrays, übergeben und gibt die resultierende Produktmatrix zurück. Sollte die Multiplikation aufgrund falscher Dimensionen nicht möglich sein, so geben Sie eine entsprechende Nachricht auf dem Bildschirm aus und liefern `null` zurück.

Hinweis: Verwenden Sie drei ineinander geschachtelte `for`-Schleifen. Die erste iteriert über die Reihen von `mat1`, die zweite iteriert über die Spalten von `mat1` und die Reihen von `mat2` und die letzte iteriert über die Spalten von `mat2`.

V11 Statischer und dynamischer Typ ★★☆

```
1  public class Alpha {
2    protected int v;
3    public Alpha(int a) {
4      v = a;  }  }
5
6  public class Beta extends Alpha {
7    public Beta(int b, int c) {
8      super(b);
9      v = c; }
10
11   public Alpha x1() {
12     super.v++;
13     return new Beta(0, v);  }
14
15   public int x2(int x) {
16     return x + ++v + v++;  }  }
17
18 public class Gamma extends Beta {
19   private short y;
20   public Gamma(int d, int e) {
21     super(d, e);
22     y = (short) d;  }
23
24   public int x2(int x) {
25     return x - y; }
26
27 public static void main(String[] args) {
28   Alpha a = new Alpha(7);
29   Beta b = new Beta(0, 1);
30   Gamma g = new Gamma(9, 2);
31   a = b.x1();
32   int t = b.x2(5);
33   a = new Beta(10, 12).x1();
34   b = g;
35   int r = g.x2(50);  }  }
```

Hinweis: Nach Zeile X heißt unmittelbar nach X, noch vor Zeile X+1.

(1) Welchen statischen und dynamischen Typ haben `a`, `b` und `g` nach Zeile 30?

(2) Welchen statischen und dynamischen Typ hat `a` und welchen Wert hat `a.v` nach Zeile 31?

(3) Welchen Wert haben `t` und `b.v` nach Zeile 32?

(4) Welchen statischen und dynamischen Typ haben `a`, `b` und welchen Wert hat `a.v` nach Zeile 34?

(5) Welchen Wert haben `r` und `b.v` nach Zeile 35?

V12 The final Countdown ★★☆

Schauen Sie folgenden Java-Code an. Beheben Sie sämtliche eingebauten Fehler, um den Code lauffähig zu machen. Was müssen Sie im Code ändern, um die folgende Ausgabe zu erhalten?

"Die Abschlussklausur von FOP ist am 9.4.2019"

```
1  public final class A {
2
3   private int value1 = 0, value2 = 0;
4   private final int value3 = 0;
5
6   private int getValue1() {
7      return value1; }
8
9   private int getValue2() {
10     return value2; }
11
12  private void setValue1(int newValue1) {
13     value1 = newValue1; }
14
15  private void setValue2(int newValue2) {
16     value2 = newValue2; }
17
18  public final void changeValue3(final int newValue3) {
19     value3 = 0; }
20 }
21
22 class B extends A {
23
24   public void changeValue3(final int newValue3) {
25     value3 = newValue3; }
26
27   public static void main(String args[]) {
28     B obj = new B();
29
30     obj.setValue1(9);
31     obj.setValue2(4);
32     obj.value3 = 2019;
33
34     String result = "Die Abschlussklausur von FOP ist am "
35     + getValue1() + "." + getValue2() + "." + obj.value3
36
37     System.out.println(result);
38     }
39 }
```

V13 Klassen, Interfaces und Methoden ★ ★ ★

V13.1

Schreiben Sie ein `public`-Interface `A` mit einer Objektmethode `m1`, die Rückgabetyp `double`, einen `int`-Parameter `n` und einen `char`-Parameter `c` hat.

V13.2

Schreiben Sie ein `public`-Interface `B`, das von `A` erbt und zusätzlich eine Objektmethode `m2` hat, die keine Parameter hat und einen `String` zurückliefert.

V13.3

Schreiben Sie eine `public`-Klasse `XY`, die `A` implementiert, aber `m1` nicht. Klasse `XY` soll ein `protected`-Attribut `p` vom Typ `long` haben sowie einen `public`-Konstruktor mit Parameter `q` vom Typ `long`. Der Konstruktor soll `p` auf den Wert von `q` setzen. Weiter soll `XY` eine `public`-Objektmethode `m3` mit Rückgabetyp `void` und Parameter `xy` vom Typ `XY` haben, aber nicht implementieren.

V13.4

Schreiben Sie eine public-Klasse `YZ`, die von `XY` erbt und `B` implementiert. Die Methode `m1` soll `n+c+p` zurückliefern und `m2` den `String` `"Hallo"`. `m3` soll den Wert `p` von `xy` auf den Wert `p` des eigenen Objektes addieren. Der Konstruktor von `YZ` ist `public`, hat einen `long`-Parameter `r` und ruft damit den Konstruktor von `XY` auf.

V14 Jedes dritte Element ★ ★ ★

Gegeben sei eine Klasse `X`. Schreiben Sie für diese Klasse die `public`-Objektmethode `foo`. Diese hat ein Array `a` von Typ `int` als formalen Parameter und liefert ein anderes Array b vom Typ `int` zurück, das aus a entsteht, indem jedes dritte Element gelöscht wird, das heißt, die Elemente von `a` an den Indizes $0, 3, 6, 9, \ldots$ werden nicht nach `b` kopiert, alle anderen Elemente von `a` werden in derselben Reihenfolge, wie sie in `a` stehen, nach `b` kopiert. Weitere Elemente hat `b` nicht. Sie dürfen voraussetzen, dass a mindestens Länge 2 hat und ungleich `null` ist. Sie dürfen einfach Operator `=` für das Kopieren von Elementen verwenden.

Hinweis: Überlegen Sie sich die Gesetzmäßigkeit, nach der die Indizes $1, 2, 4, 5, 7, 8, \ldots$ in `a` auf die Indizes $0, 1, 2, 3, 4, 5, \ldots$ in `b` abzubilden sind. Für die Länge von `b` werden Sie eine Fallunterscheidung benötigen, je nachdem, welchen Rest `a.length` dividiert durch 3 ergibt. Denken Sie auch an die letzten beiden Elemente von `a`.

H Sechste Hausübung — Gesamt 10 Punkte
Approximationen

Auf diesem Übungsblatt haben Sie verschiedene Konzepte neu kennengelernt oder vertieft. Dazu gehören Zufallszahlen, mehrdimensionale Arrays, verschachtelte Schleifen, Klassen und Interfaces. In dieser sechsten Hausübung wollen wir diese Werkzeuge nutzen, um uns dem Thema Approximation - also Näherungsverfahren - zu widmen.

Die folgenden drei Aufgaben sind allesamt unabhängig voneinander. Vergessen Sie nicht, Ihre Methoden gemäß den am Anfang des Blattes vorgestellten Konventionen zu dokumentieren!

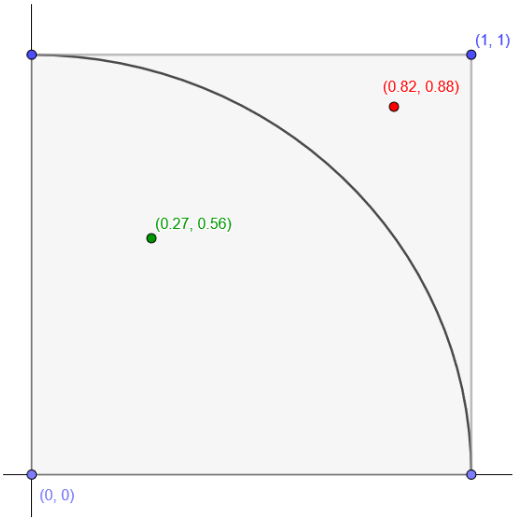
H1 Approximation von π — 2 Punkte

In der ersten Aufgabe schauen wir uns zwei verschiedene Wege zur Approximation der Kreiszahl π an. Dazu finden Sie in der Vorlage die Klasse `ApproximatePi`, in der Sie zwei Methoden zu vervollständigen haben. Beim Ausführen der Klasse `Main` wird automatisch ein Fehlerdiagramm erzeugt, welches die Güte unserer Annäherung aus den beiden Teilaufgaben visualisiert.

H1.1 Monte-Carlo-Simulation — 1 Punkt

Monte-Carlo-Simulationen gehören in die Klasse der randomisierten Algorithmen und können zur Flächeninhaltsberechnung verwendet werden. Dieses Verfahren wollen wir hier anhand der probabilistischen Berechnung der Kreiszahl π verdeutlichen.

Wir stellen uns zunächst ein Quadrat mit der Seitenlänge 1 vor. Zusätzlich betrachten wir einen Viertelkreis mit Radius 1 innerhalb dieses Quadrates. Wir erzeugen nun zufällig einen Punkt in diesem Quadrat und überprüfen, ob der Punkt innerhalb oder außerhalb des Kreises liegt.
Machen wir dies mit vielen Punkten, so gilt der Zusammenhang:



$$\frac{\pi}{4} = \frac{r^2 \cdot \pi}{(2 \cdot r)^2} \approx \frac{\text{Anzahl Punkte innerhalb des Kreises}}{\text{Anzahl aller Punkte}}$$

Betrachten Sie die Abbildung rechts für eine Darstellung des Sachverhaltes. Der zufällig erzeugte grüne Punkt ist ein Treffer innerhalb des Kreises, der rote nicht.

Vervollständigen Sie die Methode `double monteCarloPi(int n)`. Diese bekommt die Anzahl der zu generierenden Punkte `n` übergeben und liefert eine Approximation für π basierend auf dem oben beschriebenen Verfahren zurück.

H1.2 Flächeninhaltsberechnung 1 Punkt

Der exakte Flächeninhalt des Kreises kann auch über ein Integral bestimmt werden:

$$\int_0^1 \frac{4}{1+x^2}\,dx$$

Da wir dieses Integral nicht so einfach analytisch in Java lösen können, greifen wir auch hier auf eine Approximation zurück. Dafür verwenden wir hier die sogenannte numerische Trapezintegration.[1] Zur Approximation eines Integrals kann dann die folgende Formel verwendet werden:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\sum_{k=1}^{n}\left(f(x_{k-1})+f(x_k)\right) \quad \text{mit} \quad h=\frac{b-a}{n} \quad \text{und} \quad x_k = a+k\cdot h$$

Vervollständigen Sie die Methode `double integrationPi(int n)`. Diese bekommt die Schrittgröße `n` (ein größeres n steht für eine feinere Unterteilung und damit eine bessere Approximation) übergeben und gibt die Approximation des obigen Integrales zurück.

In Abbildung 1 sehen Sie die Visualisierung der Trapezintegration. Sie erstellen Trapeze unter der Kurve von f (hier grün) und berechnen deren Flächeninhalt zur Approximation.

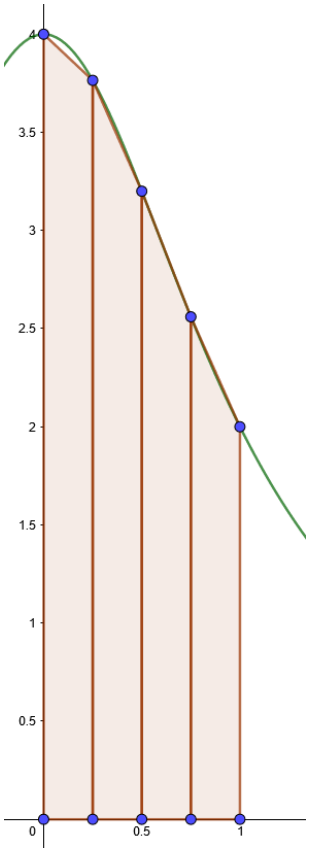


Abbildung 1: Beispiel zur Trapezintegration mit $h = 0.25$

[1] https://en.wikipedia.org/wiki/Trapezoidal_rule

H2 Bayer Pattern 3 Punkte

Jede Kamera gibt heutzutage RGB Bilder aus. Trotzdem verwenden die meisten von ihnen keine drei separaten Chips zur Farbspeicherung, sondern lediglich einen Chip, der auf einem sogenannten *Color Filter Array* basiert. An jeder Stelle im Bild wird also zunächst nur einer der drei RGB-Werte gespeichert, und anschließend werden die beiden fehlenden Werte interpoliert. Um dies zu realisieren, kann das sogenannte *Bayer Pattern* verwendet werden. Nach diesem Pattern werden die Werte folgendermaßen gespeichert: Abbildung 2 zeigt, wie die Farbwerte des Bildes abgespeichert werden und Abbildung 3 zeigt ein beispielhaftes Bild, welches in diesem Muster gespeichert wurde:

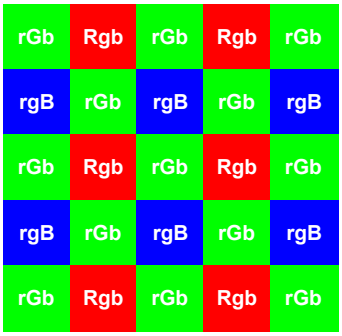


Abbildung 2: Ausschnitt des Patterns: Der jeweils gespeicherte Farbwert ist groß.

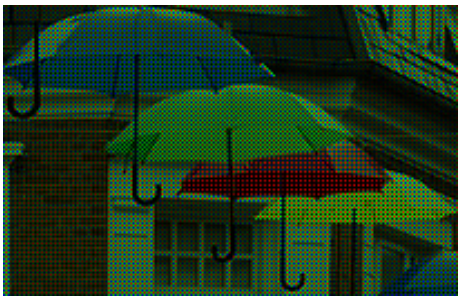


Abbildung 3: Ein Bild gespeichert im Bayer Pattern (entnommen hier).

Um aus diesem Pixel-Muster nun wieder ein normales Bild herzustellen, werden die fehlenden Werte für jeden Pixel interpoliert. Dies bedeutet Sie berechnen für einen grünen Pixel den R- und den B-Wert, für einen roten Pixel den B- und G-Wert und für einen blauen Pixel den R- und G-Wert aus den jeweiligen Werten der Nachbarschaft. Wir betrachten in dieser Aufgabe die einfachster aller Interpolationen - das arithmetische Mittel. Um also an der Stelle eines grünen Pixels den fehlenden Rotwert zu bestimmen, nehmen Sie alle direkten roten Nachbarpixel (egal ob diagonal, vertikal oder horizontal) und bilden den Mittelwert dieser Werte. Die zwei fehlenden Werte werden also immer durch den Mittelwert der Nachbarwerte geschätzt. Wenn Sie dann für jeden Pixel die zwei fehlenden Werte nach diesem Schema berechnet haben, erhalten Sie das rekonstruierte Bild. Für das Beispielbild würde damit beispielsweise Abbildung 5 entstehen:

Abbildung 4: Originaler Ausschnitt.

Abbildung 5: Rekonstruierter Ausschnitt.

H2.1 Einführung 0 Punkte

Machen Sie sich in dieser Teilaufgabe kurz mit den bereits implementierten Klassen `Image` und `BayerPattern` der Vorlage vertraut.

Die Klasse `Image` ermögtlicht es ihnen PNG-Bilder einzulesen und wieder abzuspeichern. Wir verwenden, wie auch in Hausübung 02, den RGB-Farbraum. Jedes Pixel nimmt dabei genau die Farbe an, die durch sein sogenanntes RGB-Tripel beschrieben wird. Sollten Sie hier nochmal eine Auffrischung Ihres Wissens benötigen, schauen Sie im entsprechenden Text auf Übungsblatt 02 nach.

Um die Pixel eines Bildes zu speichern, verwenden wir ein dreidimensionales `int`-Array. Der erste Index gibt an, an welcher Höhe sich ein Pixel im Bild befindet, der zweite Index gibt an, an welcher Breite sich ein Pixel im Bild befindet und der dritte Index gibt an, welcher von den drei Farbwerten (rot, grün oder blau) dort gespeichert wird. Dabei gilt für die dritten Indizes folgende Zuordnung:

`[][][0]` → rot `[][][1]` → grün `[][][2]` → blau

Somit erhält man beispielsweise den Blauwert des Pixels an der oberen linken Ecke über diese Indizes: `[0][0][2]`.

Zum Einlesen von Bildern können Sie die bereits vorhandenen Konstruktoren `Image(String filePath)` und `Image(int[][][] data)` verwenden. Der erste Konstruktor bekommt den Pfad zu einer PNG-Datei übergeben und speichert dieses Bild im Objektattribut `data` ab. Der zweite Konstruktor bekommt ein Bild als dreidimensionales `int`-Array übergeben und weist dem Objektattribut `data` das übergebene Array zu.

Die ebenfalls implementierte Methode `void saveAsPNG(String filePath)` können Sie nutzen, um ein Bild am übergebenen Dateipfad abzuspeichern.

Die Klasse `BayerPattern` ermöglicht es Ihnen über den Konstruktor `BayerPattern(String filePath)` eine Datei im Bayer Pattern zu laden. Das Bayer Pattern wird als zweidimensionals `int`-Array im Objektattribut `data` gespeichert. Auch hier beschreiben die Indizes, wie auch in der Klasse `Image`, die Position eines Pixels bzw. eines Farbwerts im Bayer Pattern. Welcher der drei Farbwerte für einen Pixel gespeichert wird, ist durch den Aufbau des Bayer Patterns (Abbildung 2) gegeben.

Die oben beschriebenen Methoden müssen Sie in den beiden Teilaufgaben verwenden, um am Ende Ihren fertigen Code auszuprobieren. In der Vorlage werden Ihnen bereits Beispieldateien mitgeliefert. Die Methoden, die Sie zu schreiben haben, befinden sich alle in der Klasse `BayerPattern` und müssen nur noch von Ihnen ergänzt werden.

H2.2 Drei getrennte Farbarrays — 1 Punkt

Ergänzen Sie zu Beginn in der Klasse `BayerPattern` die Objektmethode

`public int[][][] splitColorChannels().`

Diese benutzt das Array `int[][] data` der Klasse `BayerPattern`, welche das Bayer Pattern repräsentiert (siehe Abbildung 2). Die Methode soll nun alle Farbwerte trennen und in verschiedenen Ebenen eines neuen dreidimensionalen Arrays speichern. Ist das übergebene Array $h \times w$ groß, so ist das zurückgegebene Array $h \times w \times 3$ groß. Die letzte "Dimension" des Arrays soll dabei angeben, welcher Farbwert dort gespeichert wird. Die Beschreibung welcher Farbwert an welche Stelle geschrieben werden soll, finden Sie in obiger Teilaufgabe H2.1.

Die fehlenden Farbwerte an den jeweiligen Positionen setzen Sie auf -1.

Unverbindlicher Hinweis:

Überlegen Sie sich, welche Regelmäßigkeit Sie finden können, in welchen Mustern die Farben im Bayer Pattern auftreten. Wie können Sie beispielsweise schnell überprüfen, welche Farbe sich im Bayer Pattern an der Stelle (x, y) befindet?

H2.3 Interpolation der fehlenden Farbwerte — 2 Punkte

Ergänzen Sie nun in der Klasse `BayerPattern` die `public`-Klassenmethode

`int[][][] interpolateMissingValues(int[][][] splittedColorChannels).`

Diese bekommt ein Array übergeben, das über die Methode `int[][][] splitColorChannels()` erzeugt wurde und soll nun mittels Interpolation alle fehlenden Farbwerte berechnen (siehe Anleitung in H2) und das daraus resultierende Array zurückgeben. Im zurückgegebenen Array darf nun keine -1 als Farbwert auftreten!

Achtung mit den Pixeln am Rand! Diese haben eine andere Nachbarschaft als die innen liegenden Pixel und müssen separat von Ihnen betrachtet werden. Hier wird das arithmetische Mittel dann von der geringeren Anzahl an Nachbarn berechnet, es werden keine Pixel außerhalb des Arrays dazu gedacht oder Ähnliches.

Sollte durch die Bildung der Mittelwerte keine natürlichen Zahlen mehr auftreten, runden Sie zur **nächstkleineren Zahl** ab!

Unverbindliche Hinweise:

- Nutzen Sie die Regelmäßigkeiten im Muster, die Sie bereits in Aufgabe H2.2 erkannt und verwendet haben.
- Suchen Sie auch hier wieder eine Regelmäßigkeit, und zwar welche Nachbarschaftspixel Sie in Ihre Berechnung einbeziehen müssen. Wie können Sie schnell an alle Nachbarschaftspixel für eine gegebene Postion (x, y) kommen?

H3 Polynome — 5 Punkte

Die Approximation durch Polynome stellt in der Analysis ein mächtiges Werkzeug dar, und so wollen wir uns in dieser Aufgabe damit beschäftigen, Polynome in Java zu modellieren. Ein Polynom n-ten Grades hat die folgende funktionelle Form:

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \quad n \geq 0$$

Dabei nennen wir die Werte $a_0, \ldots, a_n$ die Koeffizienten des Polynoms. Als Grad des Polynoms bezeichnen wir den höchsten Exponenten n, für den der Koeffizient a_n im Ausdruck $a_n x^n$ nicht 0 ist (auch Leitkoeffizient genannt). Ein Polynom $n - ten$ Grades besitzt also n+1 Koeffizienten, wobei die Koeffizienten mit Ausnahme des Leitkoeffizienten auch den Wert 0 annehmen können.

Weiter ist in dieser Aufgabe nichts vorgegeben, Sie werden alles von Grund auf selbst modellieren und implementieren.

H3.1 Interface für Funktionen — 1 Punkt

Vielleicht wollen wir später auch noch andere Funktionstypen als Polynome betrachten. Daher legen wir uns zunächst ein Interface für Funktionsfamilien an. Schreiben Sie ein `public`-Interface `Function` mit einer Objektmethode `getValue`, die Rückgabetyp `double` und einen `double` Parameter hat.

H3.2 Generelle Polynome — 1 Punkt

Neben den Polynomen aus der elementaren Algebra, welche wir hier betrachten, finden Polynome auch Anwendungen in vielen anderen Bereichen. Schreiben Sie eine `public`-Klasse `GeneralPolynomial`, die `Function` implementiert, aber `getValue` nicht.

Klasse `GeneralPolynomial` soll ein `private`-Attribut `coefficients` vom Typ `double[]`, ein `private`-Attribut `degree` vom Typ `int` sowie einen `public`-Konstruktor mit Parameter `coef` vom Typ `double[]` haben. Der Konstruktor soll das Attribut `coefficients` auf `coef` setzen und `degree` auf den Grad des Polynoms. Dabei stellt das Array `coeffiecients` die Koeffizienten des Polynoms dar. Die Koeffizienten sind dabei absteigend angeordnet, für fehlende Koeffizienten wird 0 eingetragen.

Ein Beispiel:

$$s(x) = \underbrace{4x^3 - 5x^2 - 7}_{\text{Polynom 3. Grades}} \rightarrow \underbrace{\texttt{[4,-5,0,-7]}}_{\texttt{a.length = 4}}$$

Weiter soll die Klasse `GeneralPolynomial` die beiden `public`-get-Methoden `double[] getCoefficients()` und `int getDegree()` besitzen, welche die entsprechenden Attribute zurückliefern. Zusätzlich besitzt Sie die `private`-set-Methoden `void setCoefficients(double[] coefficients)` und `void setDegree(int degree)`, welche die entsprechenden Attribute setzt.

Weiter besitzt die Klasse die `public`-Objektmethoden (beide sind beide vom Rückgabetyp `GeneralPolynomial`) `firstDeriv()` und `antiDeriv()`, implementiert diese aber nicht.

H3.3 Auswertung, Ableitungen und Stammfunktionen 3 Punkte

Nun kommen wir zu den für uns interessanten Polynomen. Schreiben Sie eine public-Klasse `Polynomial`, die von `GeneralPolynomial` erbt und `Function` implementiert.

Der Konstruktor von `Polynomial` ist `public`, hat einen `double[]`-Parameter `coef` und ruft damit den Konstruktor von `GeneralPolynomial` auf.

Methode `getValue` soll das Polynom an der übergebenen Stelle `x` auswerten und die Auswertung zurückgeben.

Methode `firstDeriv` soll die erste Ableitung des Polynoms bilden und als neues Polynom zurückgeben. Die Länge des neuen Koeffizientenarrays ist dabei immer um genau 1 kleiner! Das bedeutet, sollte der vorletzte Koeffizient 0 sein, so wird im neuen Array eine 0 an letzter Stelle stehen.

Methode `antiDeriv` soll die Stammfunktion des Polynoms bilden und als neues Polynom zurückgeben. Dabei ist die Länge des neuen Arrays stehts um 1 größer und enthält an der letzten Stelle den Koeffizienten 0.

Beispiel für das Polynom $s(x) = 3x^3 + 0.333x^2 + 5$:

Der Code...

```
1 double[] coeffS = {3,0.333,0,5};
2 Polynomial s = new Polynomial(coeffS);
3 Polynomial sD = s.firstDeriv();
4 Polynomial sAD = s.antiDeriv();
5 System.out.println(Arrays.toString(s.getCoefficients()));
6 System.out.println(Arrays.toString(sD.getCoefficients()));
7 System.out.println(Arrays.toString(sAD.getCoefficients()));
8 System.out.println(s.getValue(4));
```

... liefert folgende Konsolenausgabe (inklusive Kommentare von uns):

```
1 [3.0, 0.333, 0.0, 5.0] // Koeffizienten
2 [9.0, 0.666, 0.0] // erste Ableitung
3 [0.75, 0.111, 0.0, 5.0, 0.0] // Stammfunktion
4 202.328 // Wert an der Stelle x = 4
```

Zur Erinnerung:

Die allgemeine Ableitung eines Polynoms n-ten Grades:

$$p'(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots + na_nx^{n-1} = \sum_{i=1}^{n} ia_ix^{i-1} = \sum_{i=0}^{n-1} (i+1)a_{i+1}x^i$$

Die allgemeine Stammfunktion eines Polynoms n-ten Grades:

$$\int p(x)\,\mathrm{d}x = a_0x + \frac{1}{2}a_1x^2 + \frac{1}{3}a_2x^3 + \cdots + \frac{1}{n+1}a_nx^{n+1} = \sum_{i=0}^{n} \frac{a_i}{i+1}x^{i+1} = \sum_{i=1}^{n+1} \frac{a_{i-1}}{i}x^i$$