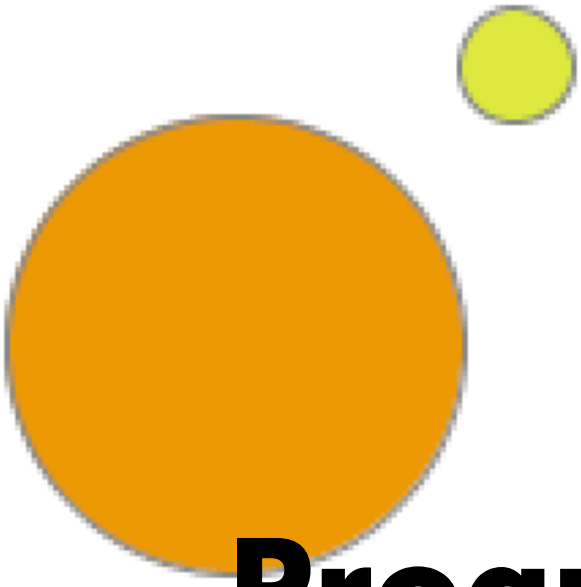


Programmieren lernen mit processing

Teil 1




Ein Arbeitsheft

Autor: Ingo Bartling

CC BY-NC-SA 4.0

Stand 9/2019

Inhaltsverzeichnis

Vorwort zur Benutzung

Dieses Arbeitsheft wurde parallel zum Unterricht im Schuljahr 2018/19 in zwei Klassen der zehnten Jahrgangsstufe entwickelt und getestet. Und es wird sich auch in den folgenden Jahren bestimmt noch weiterentwickeln.

Es ist kein Buch zum Selbstlernen. Aus meiner eigenen Erfahrung aus dem Unterricht heraus, ist es von Vorteil, wenn einem ein Lehrer die Elemente erklärt. Was den Schülern abgeht, sind dann die passenden Übungen. Frei nach der Melodie „Ich würde gerne programmieren, ich weiß aber nicht was."

Dieses Heft ist daher als ***Begleitlektüre*** zum normalen Unterricht zu verstehen und soll keinen Unterricht ersetzten, sondern begleiten. Daher gibt es auch keine Musterlösungen. Die Versuchung, einfach nur die Musterlösung zu präsentieren, wäre vielleicht doch einfach groß.

Ach ja, warum processing. Deswegen:

1. Es kostet nichts: www.processing.org.
2. Mehrere Programmiersprachen sind möglich (JAVA, Python, JavaScript) inklusiver mehrerer Plattformen.
3. Das „Hello world"-Programm ist ein 1-Zeiler: println(„Hello World");
4. Die Orientierung an Grafik ermöglicht motivierendere Ergebnisse.
5. Spiele, die immer sehr motivierend sind, sind gut umzusetzen.

Gerade die letzten beiden Punkte sind, denke ich, sehr reizvoll für die Mehrheit der Schüler. Zumindest habe ich das immer so erlebt.

Ingo Bartling (Mai 2019)

Das Heft wird im Schuljahr 2019/20 aktualisiert und erweitert. Zusätzlich erfolgt eine Zweiteilung des Heftes, da der Lehrplan des G9 eine Aufspaltung auf die 9. Klasse und 10. Klasse vorsieht. Dabei wird die Trennung ziemlich genau an der Stelle Array durchgeführt. Schwerpunkt dieses Heftes ist daher das Erstellen von Grafiken mit Hilfe von eigenen Klassen inklusive Vererbung.

Ingo Bartling (September 2019)

1.1 Was ist bedeutet Programmieren?

„Programmieren bedeutet ein Problem zu zerlegen, Algorithmen und Abläufe zu definieren und in eine Computersprache zu übersetzten."

1.2 Was ist ein Algorithmus?

Ein Algorithmus ist eine Vorschrift, die folgende Eigenschaften erfüllt:

1. **Eindeutigkeit**
   Ein Algorithmus muss in der Beschreibung eindeutig sein.

2. **Ausführbarkeit**
   Jeder Einzelschritt muss ausführbar sein.

3. **Finitheit (Endlichkeit)**
   Der Algorithmus muss in endlicher Zeit ausgeschrieben werden (und damit auch endlich lang sein)

4. **Terminierung**
   Nach endlich vielen Schritten liefert der Algorithmus immer ein Ergebnis

5. **Determiniertheit**
   Bei gleichen Startwerten kommt immer das gleiche Ergebnis heraus

6. **Determinismus**
   Zu jedem Zeitpunkt gibt es nur genaue 1 Möglichkeit, wie fortgesetzt werden kann

Aufgabe 1.2.1: Rezept

Schreibe einen Rezept aus mindestens 5 Schritten auf (oder drucke es aus und klebe es in dein Heft ein). Überprüfe nachvollziehbar anhand der obigen 6 Eigenschaften, ob ein Algorithmus vorliegt.

1.3 Was ist eine Computersprache?

Computersprache gibt es in verschiedenen Abstraktionsstufen („01010100101110“ bis Scratch) und dienen der Kommunikation mit einem Computer. Anders als sogenannte natürliche Sprachen wie Englisch, Französisch oder Deutsch sind diese Sprache vor allem eindeutig. Der Schlüsselwort „class“ oder „for“ in der Computersprache bedeutet immer das gleiche. Im Deutschen kann das durchaus anders sein. So kann „Mutter“ mal Mama bedeuten oder auch das Gegenstück bei einer Schraube sein. (Mehr dazu in der 12. Klasse)

Unsere Programmiersprache wird Java sein. Man könnte auch eine andere Sprache nehmen, da es fast egal ist mit welcher Programmiersprache man anfängt. Da aber das bayerische Abitur an Java angelehnt ist, ist dies für Schüler am sinnvollsten.

1.4 Was ist processing?

Mit Java lassen sich die unterschiedlichsten Arten von Software entwickeln: Büroanwendungen, Apps für Android-Handys, Spezial-Programme für Physik, Bank-Software, etc. Für den unerfahrenen Programmierer macht es meiner Erfahrung nach aber am meisten Freude, wenn er Interaktionen, Spiele und Grafik-Effekte erzeugen kann. Hier erzeugen selbst kleinste Anpassungen am selbst geschriebenen Quelltext sicht- und erlebbare Veränderungen.
Weitere Informationen finden sich bei processing.org.

1.5 ~~Programmieren~~ Alles lernen

Etwas Neues zu lernen erzwingt in der Regel zwei Schritte:

1. Neues integrieren
   Zunächst muss das neue Wissen in Form von Fakten gelernt werden
2. Neues festigen
   Wiederholen, wiederholen, wiederholen

Wenn ich Gitarre lernen möchte, dann muss ich zunächst die Griffe und Anschlag- oder Zupfmuster lernen. Im zweiten Schritt muss das schnelle, automatische Spielen durch viel Üben trainiert werden.

Möchte ich Portraits zeichnen, so muss ich erst die Proportionen genau lernen. Dann übe ich durch viele Wiederholungen.

Beim Programmieren lernen ist dies ähnlich. Zunächst vermittelt der Lehrer die Fakten bevor der Student oder Schüler, angeleitet durch die Lehrperson, selbstständig übt.

Beim Programmieren lernen kommt aber noch etwas anderes hinzu. Schaut man sich die Definition aus Kapitel 1.1 genau an, so habe ich bisher nur den letzten Schritt erklärt: „in eine Computersprache übersetzen".

„Programmieren bedeutet
- *ein Problem zu zerlegen,*
- *Algorithmen und Abläufe zu definieren*
- *und in eine Computersprache zu übersetzten."*

Um aber wirklich programmieren zu können, muss der Lernende sich schrittweise eine Sammlung von Algorithmen bzw. Abläufe merken, die je nach Problemstellung unterschiedlich zusammengesetzt werden. Beispiele hierfür wären „Einen Konstruktor definieren", „Alle Elemente eines Feldes ausgeben", „Eine Variable hochzählen" usw.

Das Schwierigste ist dann der letzte Schritt: „Ein Problem zu zerlegen." Aber auch hier hilft die Erfahrung und damit das Üben. Die folgende Analogie soll das verdeutlichen: Wenn du beispielsweise für deinen Freund oder Freundin ein Lied komponieren möchtest, so wird es dir nur bedingt helfen, dass du vielleicht die Akkorde C,F,G,Am auf der Gitarre kannst. Du weißt nicht in welcher Reihenfolge sie kommen sollen, oder sie passen nicht zu dem Text, den du bereits geschrieben hast. Oder die Akkorde passen gar nicht zu dem Inhalt, den du ausdrücken möchtest.

Ähnlich beim Programmieren. Du kennst vielleicht die Datentypen int, float, String und boolean. Weißt, wie man Klassen, Attribute und Methoden definiert, aber dennoch weißt du nicht, wo und wie du bei einem Programm anfangen sollst. Was kannst du dann machen?

Vorausgesetzt du kennst Programmier-Bausteine machst du genau das gleiche, wie beim Komponieren: Du fängst einfach mal an. Kein Warten, einfach mal machen und schauen, was passiert. Jede Zeile Programmcode, jeder Fehler, sofern man ihn berücksichtigt, führt dazu, dass du besser und sicherer wirst.

Learning by doing

Und das lässt sich auf alles übertragen? Ja, bis zu einem gewissen Grad. Josh Kaufman stellt das auf seiner Internetseite (first20hours.com) und seinem Buch entsprechend dar. Hier aber eine kurze Zusammenfassung:

1. **Definition und Analyse des Ziels**
   Was genau ist mein Ziel und aus welchen Teilfähigkeiten wird mein Ziel gebildet.
2. **Selbstkorrektur**
   Woran kann ich sehen, dass ich es richtig mache?
3. **Störende Elemente entfernen**
   Was könnte mich vom Lernen abhalten? Weg damit.
4. **20 Stunden üben**
   Wenn ich jeden Tag 40 Minuten konzentriert übe, sollte ich nach 1 Monat mein Ziel erreicht haben.

Beispiele, wie man die unterschiedlichsten Fähigkeiten lernen kann, findet man vielfältig bei youtube. Ein paar davon habe ich in meinem Blog zusammengefasst:

1.6 Programmieren lernen

Übertragen auf dieses Heft bzw. das Programmieren am Beispiel der Aufgabe 3.2.1 sähe das dann vielleicht so aus:

1. **Definition und Analyse des Ziels**

   Es wird processing benötigt und ich muss es benutzen können.
   Ich muss Datentypen und Klassen verstehen.
   Wissen, was Attribute und Methoden sind.
   Ich muss die wichtigsten Kommandos wie set(), draw(), ellipse() etc. kennen und benutzen könne.
   Ich muss wissen, wie man „animiert".
2. **Selbstkorrektur**
   Nachschlagewerke, Internetreferenzen und -foren. Ein Mitschüler, der es verstanden hat.
3. **Störende Elemente entfernen**
   Ich habe zu Hause einen Ort, wo ich ungestört arbeiten kann. Ich stelle mein Handy auf Flugmodus.
4. **20 Stunden programmieren oder bis man es hat**
   Ich plane mir regelmäßig Zeit fürs Programmieren ein.

KAPITEL 1.2: Einfache Datentypen und -strukturen

Ein Programm macht nur Sinn, wenn das Programm Daten hat, die verarbeitet werden. Dies können über vielfältige Arten ein Programm zugeführt werden. Die vorliegenden Daten, aber auch das Programm selbst, muss im Speicher des Computers liegen. Damit der Computer weiß, wie viel Speicher er zur Verfügung stellen muss, muss in den meisten Hochsprachen wie Java der Datentyp einer Variablen angegeben werden.

1.2.1 Primitive Datentypen

Im Themenbereich Datenbanken hast du bereits verschiedene Datentypen kennengelernt. Auch in Java gibt es sogenannten primitive Datentypen, also Datenstrukturen, die aus keinen anderen bestehen.

Aufgabe 1.2.1: Datentypen bei Datenbanken

Erstelle eine tabellarische Auflistung von mindestens 4 verschiedene Datentypen, die du beim Thema Datenbanken kennengelernt hast, gib jeweils ein Beispiel an und erkläre knapp die Datentypen.

Aufgabe 1.2.2: Datentypen in Java

Erstelle eine tabellarische Auflistung von der für uns wichtigsten primitiven Datentypen in Java. Übertrage hierzu die Tabelle in dein Heft und ergänze - mit Bleistift.

Datentyp	Beispiel	Erklärung	Wertebereich / Beispiel
boolean	true	Wahrheitswert	true, false
float	3.141f	Kommazahl	+/-1,4E-45 bis +/-3,4E+38
double	3.14	Kommazahlen	+/-4,9E-324 bis +/-1,7E+308
int	-2	ganze Zahlen	-2.147.483.648 bis 2.147.483.647
char	'a'	Ein einzelnes Zeichen	-
String	"Hallo"	Mehrere Zeichen	-

Auch wenn es kein primitiver Datentyp ist, so ist der Datentyp „String“ dennoch so wichtig, dass ich ihn als grundlegend erachte und an dieser Stelle hier einführen möchte.

Aufgabe 1.2.3: Variablen anlegen

Erläutere die Begriffe „Deklarieren“, „Initialisieren“ und „Definieren“.

Deklarieren: Es wird gesagt, dass eine Variable existieren wird. Z.B. „float preis;“

Initialisieren: Eine Variable bekommt zum ersten Mal einen Wert. „preis = 3.89;“

Definieren: Deklariern und initialsieren auf einmal. „boolean istVegan = true;“

Aufgabe 1.2.4: Variablen ausgeben

Echt primitive Datentypen werden bei der Deklaration automatisch initialisiert. Gib die zugehörigen, sogenannten Default-Werte für die Datentypen boolean, int und float an:

boolean	false
int	0
float	0.0

String ist kein primitiver Datentyp und bekommt daher keinen automatischen Wert. Der Wert ist hier anfangs: null. Das bedeutet nichts oder leer

Aufgabe 1.2.5: Superhelden-Klasse

Ziehe eine der Superhelden-Karten und wähle für jede Eigenschaft einen passenden Datentyp. Definiere sodann Variablen für die Superhelden-Eigenschaften und gib Variablenwerte wieder aus.

Aufgabe 1.2.6: Schüler-Klasse

Gib Eigenschaften, also Attribute von dir selbst so an, dass jeder wichtiger primitiver Datentyp (boolean, int, float, String) mindestens 1 mal benutzt wird und gib alle Werte wieder aus.

1.2.2 Referenz-Datentypen

Aufgabe 1.2.5: Klasse-Definition
Strukturiere die Superhelden-Attribute so, dass eine Klasse „Superheld“ entsteht.

Aufgabe 1.2.6: Methode definieren
Definiere eine Methode „ausgeben()“, die alle Attribute der Klasse Superheld ausgibt.

Aufgabe 1.2.7: Standardkonstruktor
Definiere einen Standardkonstruktor und einen Konstruktor mit Parametern, so dass alle Attribute der Klasse „Superheld“ mit Startwert belegt werden können. Achte auf eine ordentliche Strukturierung.

Aufgabe 1.2.8: Methodenkopf & -rumpf
Markiere bei den folgenden Methoden den Methodenkopf in Blau und den Methodenrumpf in Grün. Kreuze auch an.

```
void draw() {
   rect(10,10,100,200);
}
```

☒ Prozedur
☐ Funktion

```
float calculate(float x) {
     return x+3;
}
```

☐ Prozedur
☒ Funktion

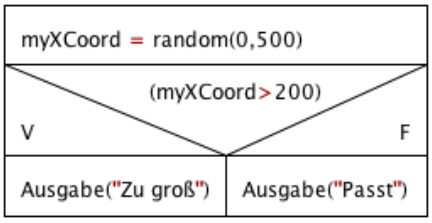
Aufgabe 1.2.9: Datenflussdiagramm
Ergänze beide „Maschinen“ so, dass die Methoden aus 1.2.8 abgebildet werden.

KAPITEL 1.3 Grundwissens-Check

- ☐ Je nach Datentyp lassen sich unterschiedlich große Informationen bzw. Arten speichern. Die wichtigsten wären für uns:
    - int – ganze Zahlen
    - float – Kommazahlen
    - String – Zeichenketten
    - boolean – Wahrheitswert
- ☐ Eine Variable muss immer einen eindeutigen Namen bekommen
- ☐ Jede Variable, wenn sie zum ersten Mal benutzt wird, muss einen eindeutigen Datentyp bekommen.
- ☐ Den Wert einer Variablen kann durch println(*variablenname*), wenn ein Zeilenumbruch (new **lin**e) erfolgen soll, oder durch print(variablenname) in der Konsole ausgegeben werden.

Aufgabe 1.4.1: Struktogramme

a) Übersetze das folgende Struktogramm in Java!




```
myXCoord = random(0,500);
if (myXCoord>200) {
        println("Zu grop");
} else {
        println("Passt");
```

b) Zeichne das Struktogramm zu folgendem Java-Quelltext!

```
if (newXCoord<0) {
    xCoord = 10;
} else {
    xCoord = newXCoord;
}
```

Aufgabe 1.4.2: Klasse mit Nicht-Standardkonstruktor

Definiere einen Klasse für Rechtecke: xCoord, yCoord, laenge, breite inkl. 2 Konstruktoren und ausgeben()-Methode.

Aufgabe 1.4.3: if-else

Definiere eine Klasse für Rechtecke: xCoord, yCoord, laenge, breite inkl. 2 Konstruktoren und ausgeben()-Methode.

Ergänze den Nicht-Standardkonstruktor um if-else-Kontrollstrukturen.

Aufgabe 1.4.4: Die Klasse Kreis

a) Definiere eine Klasse für Kreise, wobei die Klasse folgenden Attributen und Konstruktoren umfasst: xCoord, yCoord, durchmesser, inkl. 2 Konstruktoren und ausgeben()-Methode.

b) Ergänze den nicht-Standardkonstruktor um if-else-Kontrollstrukturen.

c) Verändere den Standardkonstruktor so, dass die Startwerte per Zufall belegt werden. Bediene dich dabei der processing-Funktion

```
random(float unterWert, float obererWert)
```

Beachte: Der Wertebereich von z.B. random(1,10) lautet [1;10[

KAPITAL 1.5 Was du bis jetzt können musst

Allgemeines zu Java

☐ Den Unterschied zwischen *.java und *.class-Dateien erklären können.

.java-Dateien sind das Drehbuch, .class-Dateien der Film

☐ Den Begriff kompilieren erklären können.

Durch das Kompilieren wird aus dem Drehbuch die „Filmrolle" erstellt.

☐ Erläutern können, warum ein Java-Programm kompiliert werden muss.

Nur der Film kann später von anderen auch angeschaut bzw. benutzt werden.

☐ Erläutern können, warum nicht auf jedem Computer Java-Programme ablaufen.

Da es nicht für jedes Betriebssystem einen Projektor gibt.

☐ Das Programm processing und processing.org kennen.

☐ Ein- und mehrzeilige Kommentare einfügen können und wissen, was und wie kommentiert werden muss.

Ein einzeiliger Kommentar beginnt mit //... , ein mehrzeiliger wird „geklammert" durch /*...*/ .

Es wird das Ziel jeder Methode angegeben, außer diese erschließt sich vollständig aus dem Methodennamen.

Es wird jedes Attribut und jede Variable erklärt bei der Deklaration erläutert, außer wenn sich dies vollständig aus dem Namen erschließt.

Der Methodenrumpf wird durch einzeilige Kommentare, die als Zwischenüberschriften dienen, gegliedert.

☐ Das Wort „implementieren" erläutern können.

„umsetzen", hier im Sinne von „programmieren". Wichtig für schriftliche Tests.

Die Grundbegriffe des Programmierens[1]

☐ Eine Klasse deklarieren können

class – EigenerName – { – }

☐ Attribute deklarieren können

Datentyp – eigenerName ;

☐ Eine Variable deklarieren können

Datentyp – eigenerName ;

☐ Alle wichtigen, primitive Datentypen angeben und erklären können

int (Ganze Zahlen), double (Kommazahlen: 3.14), String (Zeichenketten),
char (1 Zeichen), boolean (Wahrheitswert: true, false)

String ist genau genommen kein primitiver Datentyp, da er aus char gebildet wird. Daher auch das große S bei String.
Es gibt noch mehr primitiven Datentypen, aber diese genügen am Anfang.

☐ Eine Variable mit eine Startwert initialisieren können.

eigenerName = Wert ;

Beispiel: alter = 18; name = „Max";

Variablen, die nur innerhalb von Methoden auftauchen sollten bei der Definition bereits mit einen Startwert initialisiert werden, um Fehler zu vermeiden.
Beispiel: Datentyp – eigenerName – = – Wert ;

☐ Zeichenketten (mit Variablen-/Attributwerten) verknüpfen können

Beispiel: String antwort = „Ich heisse: " + name;

[1] in Java

☐ Grundlegende Programmierrichtlinien kennen (Einrückungen, Groß-/Kleinschreibungen, KamelHöckerschreibweise)

Alles, was Teil von etwas anderem ist, wird um 1 Tabulator eingerückt.
Klassennamen beginnen immer mit einem Großbuchstaben, Methoden (Funktionen und Prozeduren) und Attribute sowie Variablen immer mit einem Kleinbuchstaben.
Da Klassen- und Methodennamen keine Leerzeichen (keine Sonderzeichen) enthalten dürfen, aber sprechende Namen wichtig sind, beginnt jedes innere Wort eines Namens mit einem Großbuchstaben: alleDatenAusgeben()

☐ Die Aufgabe eines Konstruktors angeben können

Ein Konstruktor erzeugt gemäß des Bauplans (Klasse mit Attributen, Methoden) ein Objekt der Klasse und sollte alle Attribute initialisieren.

☐ Standardkonstruktoren .....

Ein Standardkonstruktor hat keine Parameterliste und initialisiert alle Attribute mit fest vorgegebenen Startwerten.

☐ ..... und Konstruktoren mit Parametern definieren können

Diese Konstruktoren initialisieren ebenfalls alle Attribute, allerdings benutzen sie dafür die Wert aus der Parameterliste.

☐ Die Werte der Eingangsparameter in den Attributen speichern können

attributName = parameterWert ;
Beispiel: alter = neuesAlter;

☐ Wissen, was ein Methodenkopf ...

☐ .... und was ein –rumpf ist.

```
public void alterAusgeben() {

    System.out.println(alter)
}
```

☐ Die Bestandteile eines Methodenkopfes in der richtigen Reihenfolge angeben können.

Datentyp – methodenName (ev. Parameterliste)

☐ Den Unterschied zwischen Prozedur und Funktion benennen können

Eine Prozedur liefert keine Antwort an den Aufrufenden zurück, daher ist der Rückgabewert void. Alltagsbeispiel: vonDerTafelAbschreiben() ist eine Prozedur von Schüler, da sie keine Antwort an den auffordernden Lehrer zurückgeben. Aber Sie produzieren einen Hefteintrag.
Eine Funktion gibt dem Aufrufenden eine Antwort. Die Art der Antwort (Datentyp) muss dabei vorher festgelegt werden. Alltagsbeispiel wäre bei einem Schüler die Funktion entschuldigungGeben() bei der der Schüler in Form einer Zeichenkette (String) eine Entschuldigung an den Lehrer zurückgibt.

☐ Eine Ausgabe in das Editorfenster schreiben können

Beispiel: println(„Hallo Welt“);
Ohne Zeilenumbruch am Ende: print(„Hallo Welt“);

☐ Ein Objekt im Quelltext erzeugen können

new – Konstruktorname – (– Werteliste –)
Beispiel:
Hausaufgabe neueHA = new Hausaufgabe(„M“, „20161101“, „B: S 12/1“);

☐ Ein Methode eines Objektes im Quelltext aufrufen können.

wer . was (Werteliste);
Beispiel: String alleInfos = neueHA.getInfos();
neueHA.setDate(„20161101“);

Kontrollstrukturen

☐ Eine bedingte ein/zweiseitige Verzweigung angeben können.

```
if – ( – Ja/Nein-Frage – ) – {
        //Ja-Fall
} else {
        //Nein-Fall; der nicht kommen muss
}
```

☐ Du kennst alle Vergleichsoperatoren

>, <, <=, >=, !=, ==

☐ Alle booleschen Operatoren an einem Beispiel erklären können

Und && bzw. oder || oder !

Beispiel

```
if (alter >= 16 && alter <18) {
        println(„Bier ist erlaubt.“);
}
```

KAPITEL 1.5: Graphik in Processing

Aufgabe 1.5.1: Farben

In welcher Einheit wird der Bildschirm unterteilt? Pixel

Erläutere die Bedeutung des Befehls strokeWeight(x) und fill(x). Aus welchen Wertebereich darf x gewählt werden?

strokeWeight(x) ist die Liniendicke; x>0

fill(x) ist die Füllfarbe; 0<=x<=255 in Grauwerten (Schwarz bis Weiß)

Zur Darstellung von Farben wird häufig das RGB-Modell genommen. Erläutere dies am Beispiel „fill(255,0,255)“.

Farbaddition der Farben Rot, Grün, Blau mit Werten zw. 0 und 255 (max)

fill(255,0,255) entspricht Lila

Aufgabe 1.5.2: setup() und draw()

Viele processing-Projekte werden vor allem mit Hilfe zweier Methoden gesteuert. Erläutere!

Befehl	Fragen	Deine Erläuterung

void setup() {
 	
}
	Wie oft wird diese Methode aufgerufen?
	1
		Diese Methode wird vor dem Anzeigen des Fensters einmal aufgerufen und dient der Vorbereitung des Programms.

void draw() {
 	
}
	Wie oft wird diese Methode aufgerufen?
	Unendlich
		Diese Methode wird nach dem Anzeigen des Fensters unendlich oft aufgerufen. Die Durchlaufrate pro Sekunde wird durch frameRate(x) bestimmt

Mit Hilfe welchen Befehls kann ein mehrfaches Wiederholen der Methode draw() verhindert werden? noLoop()

Aufgabe 1.5.3: Das Koordinatensystem

Erläutere das Koordinatensystem (Ursprung, Achsen) in processing!

Der Ursprung liegt oben links. Die x-Achse verläuft nach rechts.

Die y-Achse nach unten.

Aufgabe 1.5.4: Grundlagen des Zeichnens

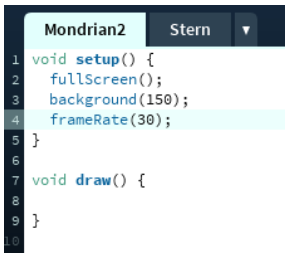
Zur Darstellung von Objekten in processing musst du ein paar wichtige Methoden und Funktionen kennen. Ergänze die Tabelle jeweils um eine knappe Erklärung oder gib den Befehl an. Beantworte auch die Fragen. Informationen findest du auf processing.org.

Befehl	Frage	Erläuterung
size(600,400)		Ein Fenster der Größe 600x400 öffnet sich.
fullScreen()		Das Zeichenfenster soll genauso groß sein wie der Bildschirm.
frameRate(60)	Wie hoch ist der Standardwert? 60	60 frames pro Sekunden, wenn möglich
background(0)		Der Bildschirm wird schwarz übermalt.
rect(100,200,300,50)	Auf welche Ecke des Objekts bezieht sich die Positionsangabe? linke, obere Ecke	Es wird ein Rechteck an der Position (100|200) mit der Breite 300 und Höhe 50 gezeichnet.
ellipse(20,50,100,100)		Ein Kreis mit M(20|50) und r=100px wird gezeichnet.
line(100,200,300,400)		Es soll eine Linie vom Punkt (100|200) zum Punkt (300|400) gezogen werden
mouseX mouseY		x-Koordinate, y-Koordinate des Mausklicks
height width		Höhe und Breite des Fensters

Kapitel 1.6: Komplexere Programmieraufgaben

1. Lese die ganze Aufgabenstellung.
2. Lies die Aufgabenstellung nochmals und markiere alle Substantive, Adjektive und Verben unter folgendem Aspekt:
   Substantiv – mögliche Klasse
   Adjektiv – mögliches Attribut einer Klasse
   Verb – mögliche Methode einer Klasse
3. Lege ein neues processing-Projekt an und definiere die Methoden setup() und draw() im ersten Reiter des Projekts.
4. Ähnlich zum Kochen erfolgt nun das sogenannte "Mise-en-place". Ergänze die setup()-Methode so, dass die äußeren Rahmenbedingungen für dein Projekt gegeben sind:
   - Fenstergröße
   - (Hintergrund)-Farbe(n)
   - frameRate(30) etc.

   Lege für jeden Referenz-Datentyp eine neue, aber noch leere Klasse in einem eigenen Reiter an.



5. Beginne nun die eigentliche Aufgabe zu lösen, in dem du möglichst einfach beginnst und schrittweise dein Programm erweiterst, verallgemeinerst und um weitere Funktionen ergänzt.

   Beachte:
   - Wechsle erst in eine neue Zeile wechselst, wenn die aktuelle Zeile korrekt ist.
   - Achte darauf, dass dein Programm immer funktioniert.

Beispiel an der Aufgabe „Mondrian 2"

1. Definiere die Methode setup() mit Methodenrumpf und draw(). Lege eine Klasse „Kreuz" an.
2. Erzeuge innerhalb der Methode draw() ein Kreuz an einer speziellen Stelle z.B. (100|100)
3. Lass dieses Kreuz an der Stelle eines Mausklicks erzeugen.
4. Passe nun die Klasse Kreuz so an, dass mit Hilfe des Standardkonstruktors „Kreuz()" ein Kreuz an der Stelle (100|100) erzeugt wird. Ergänze hierzu die Klasse um die benötigten Attribute, den Standardkonstruktor und eine Methode show(). show() ist dabei nahezu identisch zu dem Programmcode aus Punkt 2.
5. Passe nun die draw()-Methode an:

```
void draw(){
    Kreuz k1 = new Kreuz();
    k1.show();
}
```

Aufgabe 1.6.1: Abschlussaufgabe 1 (Strichcode)

a) Lege ein neues processing-Projekt mit dem Namen „Strichcode“ an.

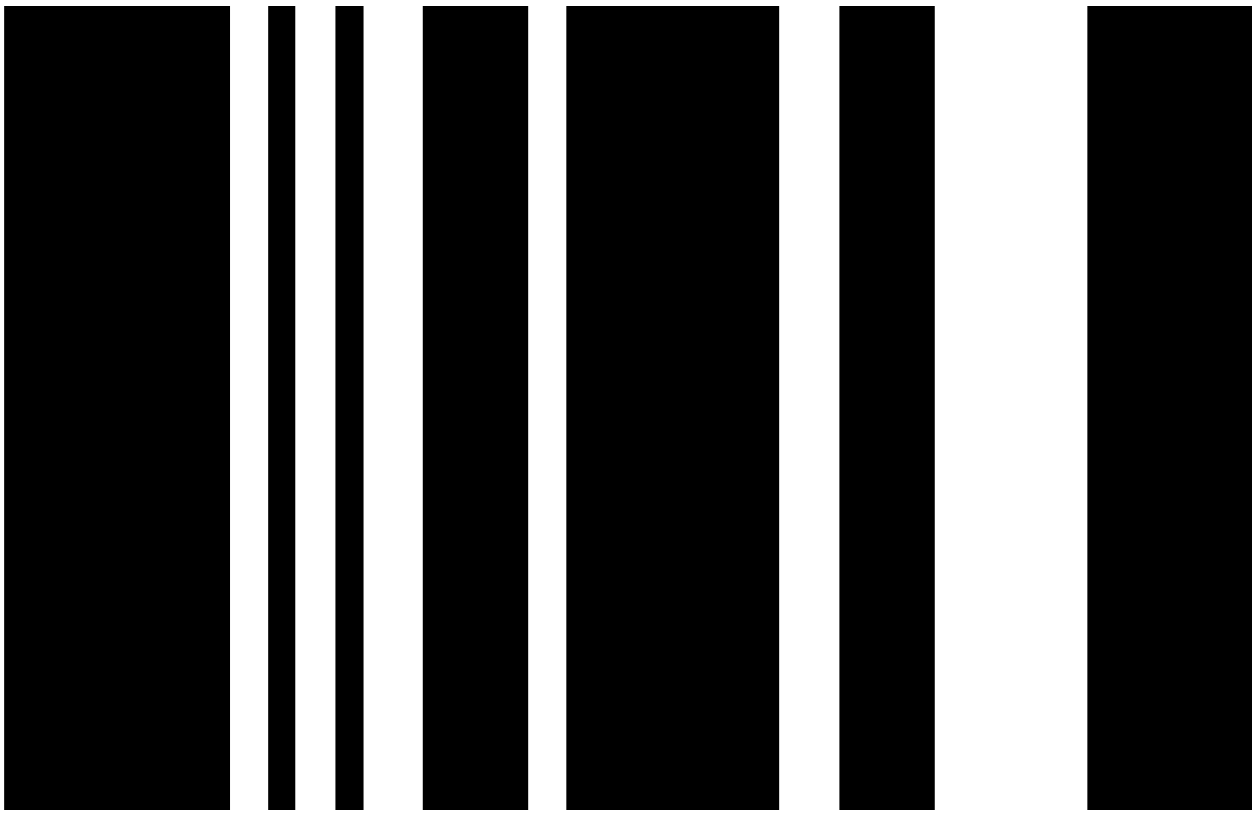
b) Erzeuge ein Fenster in Bildschirmgröße mit schwarzem Hintergrund.

c) Zeichen ein weißes Rechteck über die gesamte Bildschirmhöhe, das 100px breit ist und einen 10px breiten schwarzen Rand besitzt sowie die x-Position 50% von der Bildschirmbreite besitzt.
Warum ist das Rechteck dennoch nicht genau in der Mitte des Bildschirms?

d) Reduziere die frameRate auf 5 und lasse bei jedem neuen Bildaufbau ein bildschirmhohes, zufällig x-positioniertes, weißes Rechteck zeichnen.

e) Erhöhe die frameRate auf 10 und verändere das Programm so, dass das die Rechtecke mit Hilfe der Maus positioniert werden können, aber immer noch Bildschirm hoch sind.

f) Verändere das Programm so, dass mit 80%iger Wahrscheinlichkeit ein weißes, sonst aber ein schwarzes Rechteck gezeichnet wird.
Tipp: *random(1,10)<8*
random(1,10) liefert einen Wert aus dem Intervall [1,10[



Aufgabe 1.6.2: Abschlussaufgabe 2 (Mondrian 1)

a) Lege ein neues processing-Projekt mit dem Namen „Mondrian1“ an.

b) Erstelle ausschließlich mit den Methoden setup() und draw() ein Programm mit dem Bilder ähnlich zu Mondrians Bildern erzeugt werden können. Dabei soll ein Fenster der Größe 800x400 mit Rechtecken gefüllt werden, die eine Zufallsfüllfarbe und einen Zufallsbreite sowie –höhe haben. Die Rechtecke besitzen einen 10px breiten, schwarzen Rand.

c) Die Rechtecke sollen automatisch erzeugt werden, wobei ungefähr pro Sekunde nur 1 Rechteck erzeugt wird.

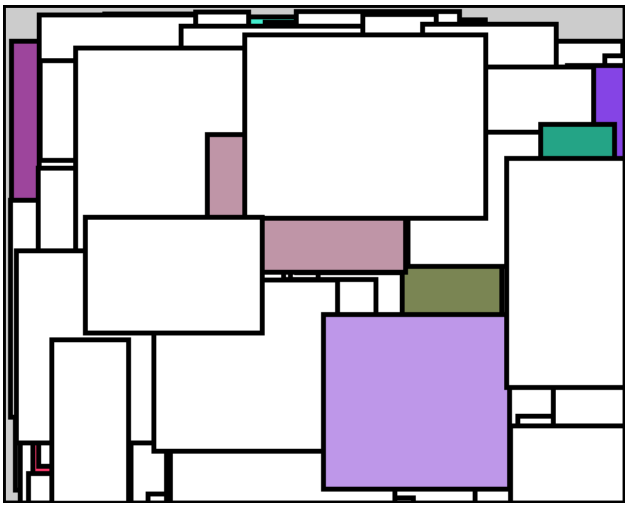
d) Durch Linksklick mit der Maus soll das Bild gelöscht werden.

e) Durch Klicken mit der linken Maustaste werden Rechtecke an der Mausposition erzeugt.

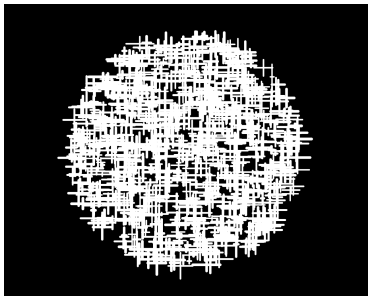
f) Statt Rechtecken werden Quadrate mit einer Zufallsgröße erzeugt.

Sternchenaufgaben dienen dazu dein Problemlösungsdenken zu testen und zu schulen. Diese Aufgaben haben ihre Schwierigkeit oftmals in der Frage: In welchen Schritten löse ich das Problem am einfachsten? Oftmals hilft es sich zu vorzustellen, wie man in der Realität das Problem lösen könnte: „Erst dann Rand malen und dann die Rechtecke. Oder erst die Rechtecke und dann zum Schluss den Rand übermalen.“

g*) Die Position der Rechtecke muss so eingeschränkt werden, dass am Rand ein 10px-breiter schwarzer Rand bleibt.



Aufgabe 1.6.3: Abschlussaufgabe 3 (Mondrian2)



Überlege zunächst welche Information du zum Zeichnen eines Kreuzes benötigst. Programmieren beginnt sollte immer mit Papier und Bleistift beginnen und nicht am Computer.

a) Lege ein neues processing-Projekt mit dem Namen „Mondrian2“ an.

b) Lege eine neue Klasse „Kreuz“ an. Jedes Kreuz soll als Attribute unter anderem xPos und yPos haben.
   Jedes Kreuz besteht aus 10px-breiten schwarzen Linien, deren Länge (horizontal wie vertikal) zwischen 10 und 50 Pixeln liegt.

c) Implementiere einen Standardkonstruktor und einen Nicht-Standardkonstruktor.

d) Implementiere auch eine Methode gibAus(), die alle Attribute-Werte in der Konsole ausgibt.

e) Ebenso wird eine Methode show() benötigt, die das Kreuz auf Basis der Attribute-Werte zeichnet.

f) Lege in der Projekt-Klasse die Methode setup() und draw() an. Bei Klick mit der linken Maustaste soll ein Kreuz gezeichnet werden.

g**) Die Position der Kreuze muss so eingeschränkt werden, dass ähnlich zu nebenstehendem Bild ein Kreis gebildet wird.

h*) Mache die Kreuze transparent. Je weiter sie von Mittelpunkt des Fensters entfernt sind, desto durchsichtiger sollen die Kreuze erscheinen. Benutze hierfür den sogenannten Alpha-Kanal einer Farbe, also z.B. stroke(255,100). 100 bestimmt in diesem Fall die Transparenz. Der Wert geht von 0 bis 255.

i**) Verändere die Transparenz oder Farbe mit der Anzahl der durchlaufenen Frames.

Benutze die processing-Funktion „line(x1,y1,x2,y2)“ mit der eine Linie vom Punkt (x1|y1) zum Punkt (x2|y2) gezogen wird.

Für die Teilaufgabe g**) kann die Funktion „dist(x1,y1,x2,y2)“ benutzt werden. Diese berechnet den Abstand zwischen dem Punkt (x1|y1) und dem Punkt (x2|y2).
Für die Teilaufgabe h*) kann zusätzlich die Funktion map() benutzt werden.
Bei der Teilaufgabe i**) kann mit dem Modulooperator % gearbeitet werden, der den Rest einer Division zurückliefert: zum Beispiel:15%256 = 15, 255%256 = 255, 256%256 = 0.

Aufgabe 1.6.4: Was du bis jetzt kennen solltest

Bei den vorangegangenen Aufgaben hast du einige neue processing-Befehle kennengelernt. Fülle die Tabelle aus, um eine Art Nachschlagewerk zu erhalten. Hilfe und weitere Informationen findest du auf processing.org.

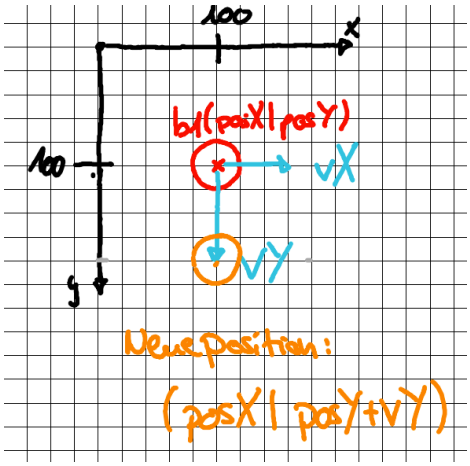
Befehl	Frage	Erläuterung
random(10)	Welchen Datentyp liefert die Funktion zurück? float	Eine Zufallszahl aus dem Bereich [0;10[
random(10,21)		Eine Zufallszahl zwischen [10,20[
int()		Wandelt einen einfachen Datentyp in eine ganze Zahl um.
int(random(1,7))	Zeichne das Datenflussdiagramm:	Es wird eine Zufallszahl zwischen 1 und 6 zurückgegeben (Würfeln)
map(mouseX,0,width,1,100)		Der Wertebereich von mouseX wird auf [1,100] abgebildet werden.
dist(x1,y1,x2,y2)		Der Abstand zwischen zwei Punkten (x1,y1) und (x2|y2)

Kapitel 2.1: Bewegung mit konstanter Geschwindigkeit

Nach den „statischen“ Effekten werden nun einzelnen Figuren wie Kreise und Linien animiert. In einem späteren Kapitel werden dann passend zu Spielen Grafiken animiert.

Anders als in der Physik, wird Bewegung damit nicht in m/s angegeben sondern in „Schrittweite in Pixel“ pro Frame. Die Bewegungsgeschwindigkeit lässt sich zum einen über die „frames per second“ steuern, aber auch über die Schrittweite pro Frame. Das Problem im ersten Fall ist, dass die framerate nicht zwingend gewährleistet werden kann, da diese auch von der Hardware abhängt und nicht beliebig nach oben geregelt werden kann. Wenn nichts anderes angegeben wurde, ist eine framerate von 30 eine gute Wahl. Die Bewegung ist flüssig und lässt sich ganz gut auf m/s umrechnen.

Im Gegensatz zur Realität ist die Bewegung in processing ruckartig: Pro Frame wird beispielsweise ein Kreis um 10px bewegt. Ist ein Kollisionsobjekt nur 1 Pixel entfernt, überlagern sich die Objekte. Wird also die „Schrittweite in Pixel“ zu hoch eingestellt, so wird diese sogenannte „collision detection“ immer komplexer und eine zugehörige immer aufwändiger. In der Regel hat sich ein Wert aus [-5;5] bis maximal |10| bewährt.



Basis der Bewegungsberechnungen bildet die Physik der (beschleunigten) Bewegung.

Im einfachsten Fall ist die Beschleunigung a=0.
Damit ergibt sich die neue y-Position des Balls aus der Berechnung:

„Alte y-Position + Geschwindigkeit“ ergibt die neue „y-Position“.

Übersetzt in Programmcode würde dies lauten:

posY = posY + vY; oder in Kurzform posY += vY;

und analog für die x-Koordinate posX += vX; .

Aufgabe 2.1.1: Klasse Ball

a) Lege ein neues processing-Projekt mit dem Namen „Ballphysik“ an. Lege einen weiteren Reiter „Ball“ in diesem Projekt an. Implementiere die Klasse „Ball“ mit einem Standardkonstruktor, so dass oben abgebildete Situation dargestellt wird: Der Ball hat einen Radius von 20px und ist an der Position (100|100).

b) Ergänze die Klasse „Ball“ um die Attribute „float vX;“ und „float vY“. Die Startwerte für die Geschwindigkeitsattribute sollen zunächst auf „vX=0“ und „vY=2;“ festgelegt sein, um eine eindeutige Ausgangssituation zu definieren.

Aufgabe 2.1.2: Bewegung mit konstanter Geschwindigkeit

a) Implementiere in der Klasse „Ball“ eine Methode „void update()“, welche die neue Position des Balls berechnet.
Auf der Karte „Ballphysik“ soll nebenstehender Inhalt dabei umgesetzt.

```
Ball b1 = new Ball();
void setup() {
  background(0);
  framerate(30);
}
void draw() {
  b1.update();
  b1.show();
}
```

b) Erläutere die Zeile

```
Ball        b1    =    new          Ball();
```

Zuweisung

Datentyp — Objektname — „ — Neues Objekt“ — Konstruktor des Datentyps

b) Erläutere weiter, warum die Zeile „Ball b1 = new Ball();“ außerhalb jeglicher Methode stehen kann.

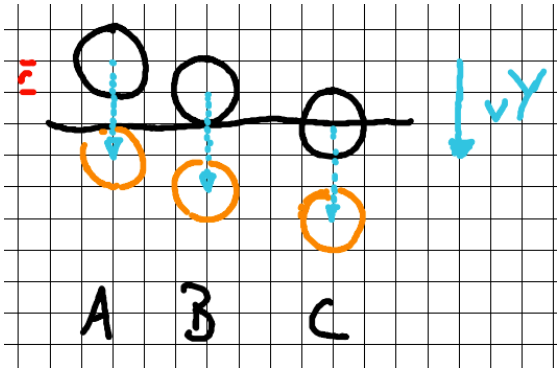
Es liegt eigentlich eine Klasse „Ballphysik“ vor. Die Zeile „class Ballphysik {}“ wird nur nicht angezeigt.

f) Welchen Effekt hätte es, wenn die Zeile „Ball b1 = new Ball();“ innerhalb der Methode draw() stehen würde.

Das Objekt b1 wird bei jedem Durchgang neu angelegt anstatt das bereits bestehende Objekt neu zu definieren(). Eine spätere Animation wäre dann nicht möglich.

g)

Damit der Ball am unteren Bildschirmrand abprallt, müssen die nebenstehenden drei Situation A, B, C erfasst werden. Hierbei wird ausgehend von der y-Position zunächst der Radius r (rot) hinzuaddiert, um den untersten Punkt des Balls zu berechnen.
Nun wird geprüft, ob beim nächsten Animationsschritt, der unterste Punkt des Balls den Bildschirmrand berührt oder sogar darüber hinzugeht.



Aufgabe 2.1.3: Abprallen am Rand

a) Setze die Formulierung des letzten Absatzes in Programmcode um:

```
if (posY + r +vY >= height) {}
```

Wenn die Bedingung für das Abprallen erfüllt ist, wird der Wert von vY einfach mit „-1“ multipliziert. Anstatt also 100+2 und damit 102 als neue y-Koordinate zu erhalten, wird 100+(-2) = 98 berechnet. Der Ball bewegt sich damit wieder nach oben.

b) Gib die vollständige bedingte Verzweigung für ein korrektes Abprall-Verhalten am unteren Bildschirmrand an.

```
if (posY + r +vY >= height) {
        vY *= -1;
}
```

c) Gib analog zur Teilaufgabe b) den Programmcode für das Abprallen am linken und rechten Bildschirmrand an.

d) Wie du vielleicht gemerkt hast, hast du zwei Mal fast den gleichen Programmcode eingegeben. Nur die Bedingung hat sich geändert. Umgangssprachlich würde man sagen: „Wenn der Ball am linken Rand ODER der Ball am rechten Rand ist, tue…“. Dieses „ODER“ gibt es auch beim Programmieren: `||` . Das logische UND würde lauten: `&&`. Fülle nun die folgenden Tabellen aus:

ODER	true	false
true	true	true
false	true	false

UND	true	false
true	true	false
false	false	false

e) Fasse die Verzweigungen nun zusammen. Eine für die Vertikal-Bewegung und eine für die Horizontal-Bewegung.

```
if (posX + r + vY >= width || posX - r + vY <= 0) {
        vX *= -1;
}
```

Kapitel 2.2: Bewegung mit konstanter Beschleunigung

Aufgabe 2.2.1: Bewegung mit konstanter Beschleunigung

Definiere in der Projekt-Klasse „Ballphysik“ eine Variable „g“als Simulation für die Erdbeschleunigung. Initialisiere g in der Methode setup() auf 0.1 . Passe nun die Geschwindigkeit innerhalb der update()-Methode an. Dabei soll vY solange um g erhöht werden, bis der maximale Wert 10 erreicht wurde, denn Körper können aufgrund der Luftreibung nicht beliebig schnell fallen.
Achte in der Methode update() darauf, dass die Schritte Beschleunigen, Abprallen, neue Position in dieser Reihenfolge abgearbeitet werden.

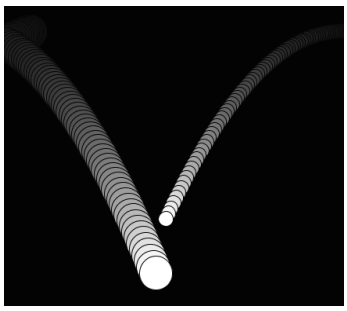
Aufgabe 2.2.2: Energieverlust

Um ein realistischeres Verhalten des Balls zu erhalten (u.a. eine abnehmende Sprunghöhe), soll noch ein „Energieverlust“ simuliert werden. Hierzu wird vX (Gleit- bzw. Rollreibung) und vY (Verformungsarbeit) bei Kollision mit dem Boden auf 90% verringert werden.
Gib die zugehörigen Programmierzeilen an:

vX = vX * 0.9;

vY *= 0.9;

Aufgabe 2.2.3: Schlieren-Effekt



Ein schöner Effekt ist der nebenstehende Effekt, den ich Schlieren-Effekt nenne. Dieser wird dadurch erreicht, dass bei jedem Durchgang von draw() das gesamte Bild leicht transparent übermalt wird:

```
//Dunkles Grau (12) das fast vollständig durchsichtig ist (10)
fill(12,10);
//Fenstergroßes Rechteck
rect(0,0,width,height);
//Füllfarbe wieder zurücksetzen
fill(255);
```

Aufgabe 2.2.4: Bildschirmschoner

Lege ein ein neues Projekt „BildschirmSchoner“ an. Erstelle zwei neue Dateien „Linie“ und „Punkt“. Der Anfangs- und Endpunkt der Linie wird durch einen „Punkt“ definiert.

a) Übertrage die Animation aus der Aufgabe 2.1.3 auf die Punkte, so dass die Punkte sich über den Bildschirm bewegen und abprallen würden.

b) Animiere nun die Linie. Benutze hierzu das beistehende Grundkonstrukt für die Klasse Linie.

```
class Linie {
  Punkt anfangsPunkt;
  Punkt endPunkt;
  float r,g,b;
  float thickness;

  void update() {
    anfangsPunkt.update();
    endPunkt.update();
  }

  void show() {
    strokeWeight(thickness);
    stroke(r,g,b);
    line(anfangsPunkt.x,anfangsPunkt.y,endPunkt.x,endPunkt.y);
  }
}
```

Aufgabe 2.2.5: Fliegender Text*

Ersetze die Linie aus Aufgabe 2.1.7 durch einen Text, der sich über den Bildschirm bewegt. Informiere dich dazu auf der Internetseite processing.org über Funktionen mit denen du die Darstellung und Bewegung des Textes steuern und beeinflussen kannst.
Hinweis: text(...), textSize(...), textWidth(...)