Prof. Dr. J. Giesl
M. Hark, J. Hensel, D. Korzeniewski

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Dienstag, den 28.11.2017 um 08:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor oder unmittelbar vor Beginn in der Globalübung abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Dienstag, dem 28.11.2017 um 08:00 Uhr an Ihre Tutorin/Ihren Tutor.
  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird und über eine geeignete `main`-Methode verfügt, um mit `java` ausgeführt zu werden. Verändern Sie außerdem den gegebenen Code nur an den angegebenen Stellen. Ansonsten werden keine Punkte vergeben.
- Benutzen Sie in ihrem Code keine Umlaute, auch nicht in Strings und Kommentaren. Diese führen oft zu Problemen, da diese Zeichen auf verschiedenen Betriebssystemen unterschiedlich behandelt werden. Dadurch kann es dazu führen, dass Ihr Programm bei Ihrer Tutorin/Ihrem Tutor bei Verwendung von Umlauten nicht mehr von `javac` akzeptiert wird.
- Für einige Programmieraufgaben benötigen Sie die Java Klasse `SimpleIO`. Diese können Sie auf der Website herunterladen.
- Halten Sie sich beim Lösen von Programmieraufgaben an die auf der Website zur Vorlesung verfügbaren Codekonventionen. Verstöße, die zu unleserlichem Code führen, können zu Punktabzug führen.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden:
  https://codescape.medien.rwth-aachen.de/progra/
  Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Tutoraufgabe 1 (Programmanalyse):

Lösen Sie die folgende Aufgabe ohne Einsatz eines Computers. Bedenken Sie, dass Sie in einer Prüfungssituation ebenfalls keinen Computer zur Verfügung haben.
Betrachten Sie das folgende Programm:

```
public class A {

    private Integer i;

    private double d;

    public A() {
        this.i = 1;
        this.d = 4;
    }

    public A(Integer x, double y) {
        this.i = x;
        this.d = y;
```

```
    }

    public A(int x, double y) {
        this.i = 3;
        this.d = x + y;
    }

    public int f(Integer x) {
        return this.i + x;
    }

    public int f(double i) {
        this.d = i;
        return this.i;
    }

    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.f(5));
        System.out.println(a1.d);
        System.out.println(a1.f(Long.valueOf(2)));
        A a2 = new A(1,1);
        System.out.println(a2.i);
        System.out.println(a2.d);
    }

}
```

Geben Sie die Ausgabe dieses Programms an. Begründen Sie Ihre Antwort.

Aufgabe 2 (Programmanalyse): (5 Punkte)

Lösen Sie die folgende Aufgabe ohne Einsatz eines Computers. Bedenken Sie, dass Sie in einer Prüfungssituation ebenfalls keinen Computer zur Verfügung haben.
Betrachten Sie das folgende Programm:

```
public class B {

    private int i1;

    private Integer i2;

    private double d1;

    private Double d2;

    public B(int a, int b, int c, int d) {
        this.i1 = b;
        this.i2 = a;
        this.d1 = d;
        this.d2 = (double)c;
    }

    public B(int a, Integer b, double c, Double d) {
        this.i1 = a;
        this.i2 = b;
```

```
        this.d1 = c;
        this.d2 = d;
    }

    public int f(long x, Float y) {
        return 11;
    }

    public int f(long x, int y) {
        return 12;
    }

    public Integer f(double x, float y) {
        return 13;
    }

    public Float g(float x) {
        return 7f;
    }

    public double g(Integer x) {
        return 8.0;
    }

    public static void main(String[] args) {
        B b1 = new B(1,2,3,4);
        System.out.println(b1.i2);
        System.out.println(b1.f(7,8));
        System.out.println(b1.f(10L,17L));
        System.out.println(b1.f(5L,6f));
        B b2 = new B(b1.i2, 5, 6, 9);
        System.out.println(b2.d2);
        System.out.println(b2.f(b1.d1,b1.i2));
        B b3 = new B(b2.i1, 14, 1.5, 16.);
        System.out.println(b3.d1);
        System.out.println(b3.g(b1.i1));
        System.out.println(b3.g(Integer.valueOf(18)));
        System.out.println(b3.f(19, b2.g(21)));
    }

}
```

Geben Sie die Ausgabe dieses Programms an. Begründen Sie Ihre Antwort.

Tutoraufgabe 3 (Rekursion):

Betrachten Sie folgende Methode:

```
public static int arraySum(int[] a) {
    int res = 0;
    for (int i = 0; i < a.length; i++) {
        res += a[i];
    }
    return res;
}
```

Schreiben Sie eine statische Methode `arraySumRecursive`, welche ein `int`-Array erhält und eine `int`-Zahl zurückliefert, sodass für jedes `int`-Array `a` die Aufrufe `arraySum(a)` und `arraySumRecursive(a)` das gleiche Ergebnis liefern. Sie dürfen in dieser Aufgabe keine Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt (inklusive der Definition von Hilfsmethoden).

Hinweise:

- Wenn Sie Ihr Programm mit der Anweisung `import java.util.Arrays;` beginnen, können Sie die vordefinierte statische Methode `Arrays.copyOfRange` benutzen. Hierbei liefert `Arrays.copyOfRange(a, i, j)` ein neues Array mit den Werten `a[i], a[i+1],..., a[j]` zurück.

Aufgabe 4 (Rekursion): (5 Punkte)

Betrachten Sie folgende Methode aus der Klasse `InsertionSort`, die auf der Webseite zur Verfügung steht:

```
public static int[] sortI(int[] arr) {
  for(int upto = 1; upto < arr.length; ++upto) {
    for(int current = upto; current > 0; --current) {
      if(arr[current] >= arr[current-1]) {
        break;
      } else {
        //aktuelles Element gehoert weiter vorne einsortiert
        int tmp = arr[current];
        arr[current] = arr[current-1];
        arr[current-1] = tmp;
      }
    }
  }
  return arr;
}
```

Die Methode `sortI` sortiert ein gegebenes Array mit dem Algorithmus *Insertion Sort*. Dieser Algorithmus funktioniert wie folgt: Zunächst wird das zweite Element betrachtet. Dieses Element wird so lange eine Position nach vorne verschoben, bis davor ein kleineres Element steht. Anschließend wird das gleiche mit dem dritten Element gemacht und so weiter. Somit ist der vordere Teil des Arrays, bis zum Index `upto`, bereits sortiert, abgesehen vom Element `current`, das noch an seine Position geschoben werden muss. Nachdem das letzte Element an seine Position geschoben wurde, ist das Array komplett sortiert.
Als Beispiel betrachten wir das Array `{5, 3, 0}`. Das Element am Index `upto = 1` ist `3`. Dieses wird mit den Elementen davor verglichen und an die richtige Position verschoben, was `{3, 5, 0}` ergibt. Nun ist `upto = 2` und die Elemente vor dem entsprechenden Element `0` sind schon sortiert. Das Element `0` muss nach vorne verschoben werden, wodurch sich `{0, 3, 5}` ergibt.

Implementieren Sie die Methode `sortR`, die den gleichen Algorithmus umsetzt, aber ***rekursiv*** arbeitet.
Sie dürfen keine vordefinierten Methoden oder Objekte in Ihrem Code verwenden.
In dieser Aufgabe dürfen Sie **keine Schleifen** verwenden. Die Verwendung von Rekursion ist hingegen erlaubt (inklusive der Definition von Hilfsmethoden).

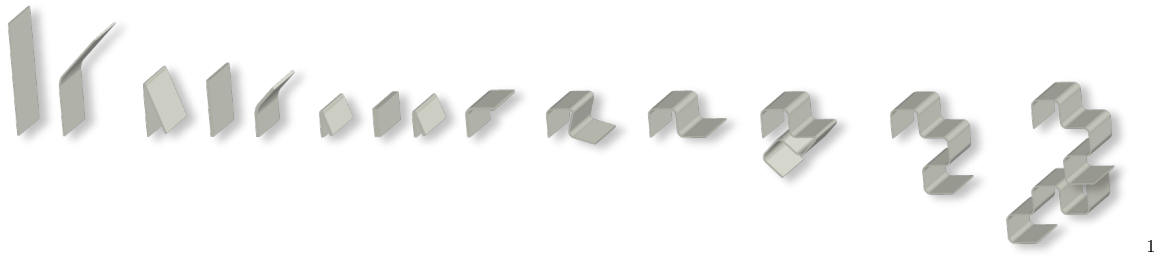
Die Klasse `InsertionSort` enthält eine `main`-Methode, die Ihre Methode an mehreren, zufällig generierten Arrays testet. In der Ausgabe sollten die 1. und 2., sowie 3. und 4., etc. Zeile jeweils gleich sein. Die Ausgabe könnte also so aussehen:

```
$ java InsertionSort
[]
[]
[-3]
[-3]
[-7, 1, 7]
[-7, 1, 7]
[]
```

```
[]
[-6, -4, -4, 0, 1, 8, 9, 11]
[-6, -4, -4, 0, 1, 8, 9, 11]
[10]
[10]
[-6, -5, -4, -1, 6, 11, 11]
[-6, -5, -4, -1, 6, 11, 11]
[-7, -6, -3, -2, -2, -1, 1, 1, 2, 3, 5, 6, 6, 9, 9, 11, 11]
[-7, -6, -3, -2, -2, -1, 1, 1, 2, 3, 5, 6, 6, 9, 9, 11, 11]
[-7, -7, -6, -5, -4, 1, 2, 3, 5, 5, 6, 6, 8, 9, 10, 10, 10]
[-7, -7, -6, -5, -4, 1, 2, 3, 5, 5, 6, 6, 8, 9, 10, 10, 10]
[-5, -4, -4, -3, 5, 6, 11, 11]
[-5, -4, -4, -3, 5, 6, 11, 11]
[-7, -6, -5, -4, -4, -3, -3, -2, 0, 0, 1, 5, 5, 7, 7, 8, 11]
[-7, -6, -5, -4, -4, -3, -3, -2, 0, 0, 1, 5, 5, 7, 7, 8, 11]
[-6, -5, -5, -4, -3, 0, 1, 2, 3, 5, 6, 8, 11, 12, 12]
[-6, -5, -5, -4, -3, 0, 1, 2, 3, 5, 6, 8, 11, 12, 12]
All tests passed.
```

Tutoraufgabe 5 (Drachenkurve):

Ziel dieser Aufgabe ist es, ein Programm zur Visualisierung der Drachenkurve zu schreiben. Die Drachenkurve erhält man, indem man einen Papierstreifen wie folgt faltet: Man falte die beiden Enden des Papierstreifens so aufeinander, dass sich die Länge des Streifens halbiert. Den so verkürzten Streifen faltet man wieder auf die selbe Art (und—wichtig—in dieselbe Richtung). Nach ein paar Wiederholungen faltet man den Streifen wieder auseinander und richtet ihn dann so aus, dass jeder Knick genau rechtwinklig verläuft. Hat man den Streifen anfänglich n-mal halbiert, erhält man durch diese Anleitung eine Drachenkurve der n-ten Ordnung.



[1]

Diese Drachenkurve n-ter Ordnung lässt sich auch durch ihre Folge von **R**echts- bzw. **L**inksknicken beschreiben:

Ordnung	Folge
1	R
2	R **R** *L*
3	R R L **R** R *L* L
4	R R L R R L L **R** R R L *L* R L L
...	...

Hierbei ergibt sich folgende Konstruktionsvorschrift: Eine Drachenkurve der 1-ten Ordnung hat nur einen **R**echtsknick. Um eine Drachenkurve der Ordnung $(n+1)$ zu erhalten, führt man erst die Folge von Knicken einer Kurve der Ordnung n durch, dann einen **R**echtsknick und dann wieder die Knicke einer Folge der Ordnung n, wobei man in deren Mitte anstelle eines **R**echtsknicks einen **L**inksknick macht.
Für Ihre Implementierung benötigen Sie die Datei `Canvas.java` von der Homepage. Verwenden Sie das Programm Javadoc, um die Schnittstellendokumentation dieser Klasse zu erzeugen.
Ergänzen Sie die Datei `Drachenkurve.java` um zwei statische Methoden `kurveL` und `kurveR`, die jeweils eine Drachenkurve beliebiger Ordnung mit einem Links- bzw. Rechtsknick in der Mitte darstellen. Diese Methoden

[1] Bild lizenziert unter CC BY-SA 3.0, Quelle:
`http://de.wikipedia.org/w/index.php?title=Datei:Dragon_curve_paper_strip.png`

bekommen als Argumente jeweils ein Objekt der Klasse `Canvas` sowie eine Ordnung als `int`-Wert. Mit Hilfe der Methoden des `Canvas`-Objektes kann die Zeichnung gefertigt werden.

Hinweise:

- Verwenden Sie die Methode `drawForward` der Klasse `Canvas` um einen Strich zu malen. Als Länge eignen sich 10 Pixel.
- Rotationen können Sie mit der Methode `rotate` der Klasse `Canvas` erreichen, die eine Gradzahl als Parameter bekommt.
- Implementieren Sie die Methoden `kurveL` und `kurveR` rekursiv.

Aufgabe 6 (Gosper-Kurve): (5 Punkte)

Auch in dieser Aufgabe soll eine fraktale Struktur mit Hilfe der Klasse `Canvas` gezeichnet werden. Diesmal geht es um Gosper-Kurven. Eine solche Kurve 0. Ordnung besteht aus einer geraden Linie. Kurven n-ter Ordnung für $n > 0$ werden gebildet, indem man sieben Gosper-Kurven $(n-1)$-ter Ordnung kombiniert. Dabei unterscheidet man zwischen Links-Kurven und Rechts-Kurven.
Eine Gosper-Links-Kurve n-ter Ordnung wird wie folgt konstruiert:

- Zunächst wird eine Gosper-Links-Kurve $(n-1)$-ter Ordnung gezeichnet
- Dann ein $-60°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Ein $-120°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Ein $60°$ Knick
- Eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $120°$ Knick
- Zwei Gosper-Links-Kurven $(n-1)$-ter Ordnung
- Ein $60°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Die nächste Kurve setzt dann im Winkel von $-60°$ an.

Gosper-Rechts-Kurven n-ter Ordnung werden ähnlich erzeugt:

- Nach der vorhergehenden Kurve wird $60°$ nach rechts angesetzt.
- Dann folgt eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $-60°$ Knick
- Zwei Gosper-Rechts-Kurven $(n-1)$-ter Ordnung
- Ein $-120°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Ein $-60°$ Knick
- Eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $120°$ Knick

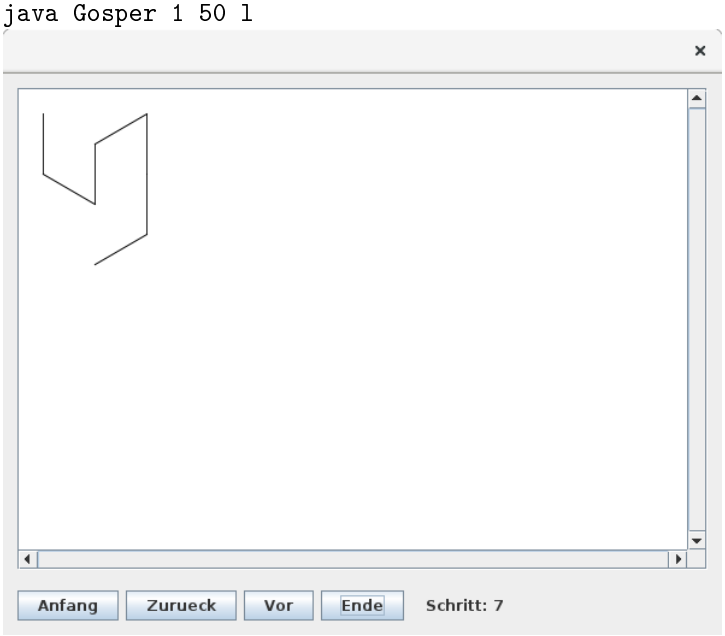
- Eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein 60° Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung

Positive Winkel bedeuten eine Ecke im Uhrzeigersinn, negative Winkel eine Ecke entgegen dem Uhrzeigersinn. Wie in der vorigen Aufgabe dürfen Sie in dieser Aufgabe keine Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.
Implementieren Sie die statischen Methoden `gosperLinks` und `gosperRechts` in der Klasse `Gosper`, welche folgende Parameter erhalten:

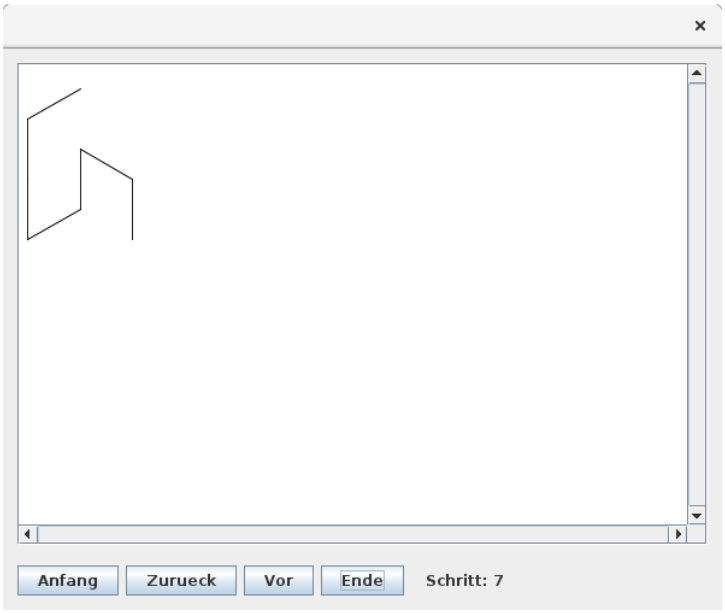
- eine Referenz `c` auf ein `Canvas` Objekt
- eine `int`-Zahl, welche die gewünschte Ordnung der Kurve angibt.
- eine `int`-Zahl, welche die Länge einer Gosper-Kurve 0. Ordnung angibt.

Diese Methode soll eine Gosper-Kurve der spezifizierten Ordnung zeichnen.
Zum Testen Ihrer Implementierung enthält die Klasse `Gosper` schon eine `main`-Methode. Das Programm bekommt bis zu drei Parameter. Der erste gibt die Ordnung der Kurve an, der zweite die Länge der Kurven 0. Ordnung aus denen sie zusammengesetzt werden soll. Der dritte Parameter gibt an, ob es eine Links-Kurve (`l`) oder eine Rechts-Kurve (`r`) sein soll. Aus der `main`-Methode wird die Methode `gosperLinks` bzw. `gosperRechts` entsprechend aufgerufen. Sie können Ihre Implementierung mit folgenden Aufrufen testen (darunter finden Sie Abbildungen, die Sie als Ergebnis zu diesen Aufrufen erhalten sollten):

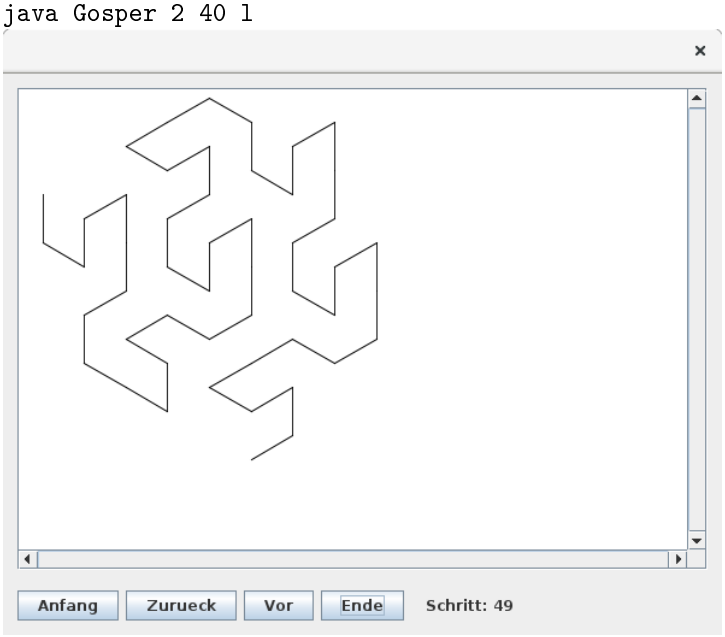
- `java Gosper 1 50 l`



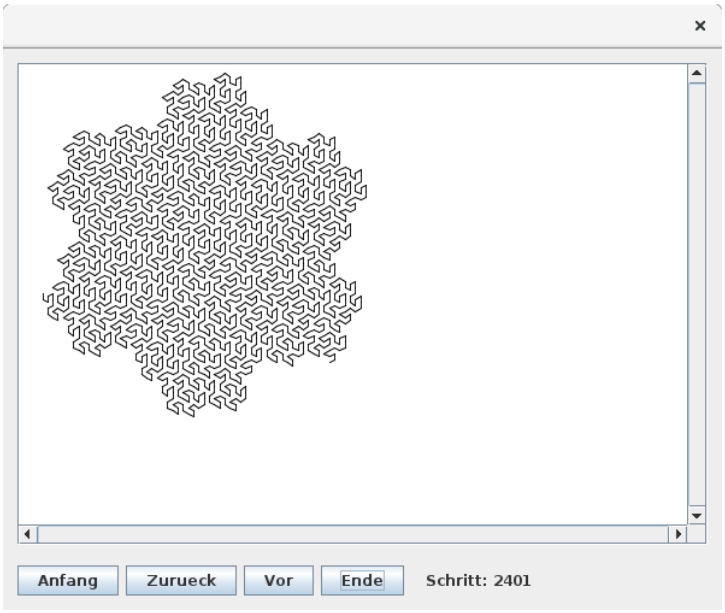
- `java Gosper 1 50 r`



- `java Gosper 2 40 l`



- `java Gosper`



Hinweise:

- Klicken Sie einmal auf die Schaltfläche Ende, um das Ergebnis anzuzeigen.
- Mit den Schaltflächen Vor und Zurueck können Sie die Zeichnung Schrittweise auf- bzw. abbauen. Der Ablauf entspricht dabei dem Ablauf Ihres Programms. Die Schaltflächen Anfang und Ende springen zum Anfang bzw. Ende des Ablaufs.

Aufgabe 7 (Deck 4): (Codescape)

Lösen Sie die Räume von Deck 4 des Spiels Codescape.