```java
//@author Maximlian Raspe
public class TurtleState {
    private double x;
    private double y;
    private double angle;
    private boolean down;
    public double getX() {
        return this.x;
    }
    public double getY() {
        return this.y;
    }
    public double getAngle() {
        return this.angle;
    }
    public boolean getDown() {
        return this.down;
    }
    public void setAngle(double angle) {
        this.angle = angle;
    }
    public void setDown(boolean down) {
        this.down = down;
    }
    public void setX(double x) {
        this.x = x;
    }
    public void setY(double y) {
        this.y = y;
    }
}

public class Wuerfel {
    public static void main(String[] args) {
        wuerfel();
    }
    public static void wuerfel() {
        Turtle wurfel = new Turtle(new LineFrame("Würfel"));
        wurfel.penUp(); // zur mitte der lineframe springen
        wurfel.turn(-Math.PI / 4);
        wurfel.forward(0.5);
        wurfel.penDown();
        wurfel.turn(Math.PI / 4);
        wurfel.forward(0.2);
        wurfel.turn(-Math.PI / 2);
        wurfel.forward(0.2);
        wurfel.turn(-Math.PI / 2);
        wurfel.forward(0.2);
        wurfel.turn(-Math.PI / 2);
        wurfel.forward(0.2);
        wurfel.turn(-Math.PI / 4);
        wurfel.forward(0.1);
        wurfel.turn(-Math.PI / 4);
        wurfel.forward(0.2);
        wurfel.turn(5*Math.PI / 4);
        wurfel.forward(0.1);
        wurfel.turn(Math.PI / 4);
        wurfel.forward(0.2);
        wurfel.turn(3 * Math.PI / 4);
        wurfel.forward(0.1);
        wurfel.turn(Math.PI / 4);
        wurfel.forward(0.2);
        wurfel.penUp();
```

```
    }
}

//@author Maximilian Raspe
public class Turtle {
    private TurtleState state;
    private LineFrame draw;
    /**
     * Konstruktor der nur das LineFrame übergeben bekommt und den anderen
     * Konstruktor mit default-Werten aufruft.
     *
     * @param lineFrame
     *            Das Visualisierungsfenster, auf dem gezeichnet wird.
     */
    public Turtle(LineFrame lineFrame) {
        this(lineFrame, 0.5, 0.5, Math.PI / 2);
    }
    /**
     * Konstruktor, der Werte übergeben bekommt und diese den internen Feldern
     * zuweist.
     *
     * @param lineFrame
     *            Das Visualisierungsfenster, auf dem gezeichnet wird.
     * @param initX
     *            x-Position der Turtle
     * @param initY
     *            y-Position der Turtle
     * @param initAngle
     *            Ausrichtung (Winkel) der Turtle
     */
    public Turtle(LineFrame lineFrame, double initX, double initY, double
initAngle) {
       draw = lineFrame;
       state = new TurtleState();
    }
    /**
     * Setzt den Stift auf dem Blatt ab.
     */
    public void penDown() {
        state.setDown(true);    //true = stift aufgesetzt, getDown um wert zu
erfassen
    }
    /**
     * Hebt den Stift vom Blatt weg.
     */
    public void penUp() {
        state.setDown(false);   //false = stift oben
    }
    /**
     * Bewegt die Schildkröte um stepSize in die blickrichtung der Schildkröte.
     * StepSize hat dabei einen Wert zwischen 0 und 1, wobei 1 gleich einer
     * Bildschrimlänge entspricht.
     *
     * @param stepSize
     */
    public void forward(double stepSize) {
            double x0 = state.getX();
            double y0 = state.getY();
            double x1 = x0 + stepSize * Math.cos(state.getAngle());
            double y1 = y0 + stepSize * Math.sin(-state.getAngle());
            state.setX(x1);
            state.setY(y1);
            if(state.getDown() == true) {  //wenn stift unten, dann zeichnen
```

```
                draw.drawLine(x0, y0, x1, y1);
            }
        }
        /**
         * Dreht sich um den angegebenen Wert.
         *
         * @param angle
         *            Der Wert um den sich die Turtle dreht.
         */
        public void turn(double angle) {
            state.setAngle(state.getAngle() + angle);
        }
}

//@author maximilian raspe
import java.util.Scanner;
public class Kochkurve {
    public static void main(String [] args) {
        Scanner stufe = new Scanner(System.in);
        System.out.println("Anzahl der Iterationen eingeben");
        int iterationen = stufe.nextInt();
        stufe.close();
        Turtle kochkurve = new Turtle(new LineFrame("Kochkurve"));
        double frameLaenge = 1.0;
        double kantenlaenge = 0.7; // Kanten länge.
        double hoehe = (kantenlaenge / 2) * Math.sqrt(3); // Höhe des Dreiecks.
        double x1 = frameLaenge / 2; // x- Koordinaten für die erste Ecke (Oben
mitte).
        double y1 = (frameLaenge - hoehe) / 2; // y-Koordinaten für die erste
Ecke (Oben mitte).
        // Bereite Startpunkt und Winkel der Turtle vor
        double dx = x1 - 0;
        double dy = y1 - 0;
        double startAngle = Math.atan(dy / dx);
        double help = Math.pow(dy, 2) + Math.pow(dx, 2);
        double startStepSize = Math.sqrt(help);
        kochkurve.turn(-startAngle);
        kochkurve.forward(startStepSize);
        kochkurve.turn(-((Math.PI / 6) + (Math.PI / 2) - startAngle));
        kochkurve.penDown();
        zeichnen(kochkurve,iterationen, kantenlaenge);
        kochkurve.turn(Math.PI / 1.5);
        zeichnen(kochkurve, iterationen, kantenlaenge);
        kochkurve.turn(Math.PI / 1.5);
        zeichnen(kochkurve, iterationen, kantenlaenge);
        kochkurve.turn(Math.PI / 1.5);
    }
    public static void zeichnen(Turtle kochkurve, int iterationen,double
kantenlaenge ) {
        if(iterationen == 0) {
            kochkurve.penDown();
            kochkurve.forward(kantenlaenge);
        }
        else {
            zeichnen(kochkurve, iterationen - 1, kantenlaenge / 3);
            kochkurve.turn(-Math.PI / 3);
            zeichnen(kochkurve, iterationen - 1, kantenlaenge / 3);
            kochkurve.turn(Math.PI / 1.5);
            zeichnen(kochkurve, iterationen - 1, kantenlaenge / 3);
            kochkurve.turn(-Math.PI / 3);
            zeichnen(kochkurve, iterationen - 1, kantenlaenge / 3);
        }
    }
```

```java
}

//@author maximilian raspe
public class Auto {
    private int speed;
    public int getSpeed() {
        return this.speed;
    }
    public void setSpeed(int speed) {
        this.speed = speed;
    }
    public Auto(int maxSpeed) {
        this.speed = maxSpeed;
    }
    @Override
    public String toString() {
        return Integer.toString(speed);
    }
}

//@author maximilian raspe
public class Fahrbahn {
    private Auto[] autosZellen;
    private int anzahlZellen;
    private int maxGeschwindigkeit;
    private double troedel;
    /**
     * Konstruktor, der einen Array der Zellen Größe erstellt von typ Klasse
Auto,
     * und rechnet auf Basis der Eingaben wie Groß der Abstand zwischen zwei
Autos
     * ist.
     *
     * @param zellen             wie Groß die Bahn ist.
     * @param anzahlAutos        wie viele Autos haben wir als Eingabe bekommen.
     * @param maxGeschwindigkeit Maximale Geschwindigkeit als Eingabe.
     * @param troedel            Trödelwahrscheinlichkeit als Eingabe.
     */
    public Fahrbahn(int zellen, int anzahlAutos, int maxGeschwindigkeit, double
troedel) {
        this.anzahlZellen = zellen;
        this.maxGeschwindigkeit = maxGeschwindigkeit;
        this.troedel = troedel;
        autosZellen = new Auto[zellen];
        double div = (double) zellen / anzahlAutos;
        int anzahlAutosGrosserAbstand = zellen % anzahlAutos;
        int anzahlAutosKleinerAbstand = anzahlAutos - anzahlAutosGrosserAbstand;
        int abstandAbgerundet = (int) Math.floor(div);
        int abstandAufgerundet = (int) Math.ceil(div);
        int index = 0;
        for (int i = 0; i < anzahlAutosGrosserAbstand; i++) {
            autosZellen[index] = new Auto(maxGeschwindigkeit);
            index += abstandAufgerundet;
        }
        for (int i = 0; i < anzahlAutosKleinerAbstand; i++) {
            autosZellen[index] = new Auto(maxGeschwindigkeit);
            index += abstandAbgerundet;
        }
    }
    /**
     * Zeichnet die Zellen. Zahlen(Geschwindigkeit): für die Plätze, in denen es
ein
     * Auto gibt. Unterstrich: für die Plätze, in denen es kein Auto gibt.
```

```
     */
    public void ausgeben() {
        for (int i = 0; i < autosZellen.length; i++) {
            if (autosZellen[i] == null) {
                System.out.print("_ ");
            } else {
                System.out.print(autosZellen[i] + " ");
            }
        }
        System.out.println();
    }
    /**
     * rechnet der Abstand zwischen zwei Autos.
     *
     * @return count: anzahl Zellen zwischen zwei Autos.
     */
    public int abstand2Autos(int autoIndex) {
        int count = 0;
        int i = (autoIndex + 1) % anzahlZellen;
        while (autosZellen[i] == null) {
            count++;
            i = (i + 1) % anzahlZellen;
        }
        return count;
    }
    /**
     * Die Regeln.
     */
    public void update() {
        int i = 0;
        while (i < anzahlZellen) {
            Auto auto = autosZellen[i];
            if (auto == null) {
                i++;
                continue;
            }
            // Regel 1 erh?he Geschwindigkeit
            int aktuelleGeschwindigkeit = auto.getSpeed();
            if (aktuelleGeschwindigkeit < maxGeschwindigkeit) {
                auto.setSpeed(++aktuelleGeschwindigkeit);
            }
            // Regel 2
            int anzahlFreieZellen = abstand2Autos(i);
            if (aktuelleGeschwindigkeit > anzahlFreieZellen) {
                aktuelleGeschwindigkeit = anzahlFreieZellen;
                auto.setSpeed(aktuelleGeschwindigkeit);
            }
            // Regel 3
            double random = Math.random();
            if (random < troedel && aktuelleGeschwindigkeit > 0) {
                auto.setSpeed(--aktuelleGeschwindigkeit);
            }
            // Regel 4
            int neuePosition = (i + aktuelleGeschwindigkeit) % anzahlZellen;
            autosZellen[neuePosition] = autosZellen[i];
            if (aktuelleGeschwindigkeit != 0) {
                autosZellen[i] = null;
            }
            i += (aktuelleGeschwindigkeit + 1);
        }
    }
}
```

```java
//@author maximilian raspe
import java.util.Scanner;
public class NaSchModell {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Fahrbahngröße eingeben");
        int fahrbahngroesse = scan.nextInt();
        System.out.println("Fahrzeug Anzahl eingeben");
        int fahrzeuganzahl = scan.nextInt();
        System.out.println("Maximalgeschwindigkeit eingeben");
        int maxgeschwindigkeit = scan.nextInt();
        System.out.println("Trödelwahrscheinlichkeit zwischen 0 und 1
eingeben");
        double wahrscheinlichkeit = scan.nextDouble();
        System.out.println("Updateanzahl eingeben");
        int updatezahl = scan.nextInt();
        scan.close();
        Fahrbahn strecke = new Fahrbahn(fahrbahngroesse, fahrzeuganzahl,
maxgeschwindigkeit, wahrscheinlichkeit);
        // Initialausgabe
        strecke.ausgeben();
        // Updates
        for (int i = 1; i < updatezahl; i++) {
            strecke.update();
            strecke.ausgeben();
        }
    }
}
```