Jaka Černetič

G 4. A

RSO

|  |
| --- |
| ***Teme:***  *Rekurzivni postopki (utrjevanje)*  *Izbrani postopki* |

|  |
| --- |
| **Dijaki pri izvajanju izberejo 5 nalog in jih rešijo.**  Opomba pred začetkom:  (dokument kot vir):  viri/900\_Karatsuba\_Offman.pdf  (risanje po Canvas-u, okvir za preprosto animacijo):  01\_04\_RSO04\_vaja\_KochTest  [Working with Canvas | JavaFX 2 Tutorials and Documentation (oracle.com)](https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm), apr. 2013, dosegljivo še sept. 2025 |

**Naloga 1**

Napišite rekurzivno realizacijo izvedbe postopka množenja velikih števil. Postopek je opisan v relevantnem dokumentu, lahko pa si ga ogledate tudi na internetu. Predpostavite, da boste med seboj množili 2 vsaj 2 mestni števili. Predlagam, da ugotovite, kako velika števila lahko zmnožite z vgrajenimi tipi. Zapis osnovnega števila je konec koncev lahko tudi (zelo dolg) niz.

**Naloga 2**

Sestavite FX javanski program, ki bo izrisoval Kochov fraktal,. Koda naj implementira DF postopek (diskusija o teh vrstah postopkov poteka/bo pri urah teorije) : najprej naj izračuna vse potrebne točke in nato izriše zahtevan fraktal.

Izvajanje postopka na vsaki stopnji izvajanja naj poteka kot:

1. daljico razdelimo na tri enako dolge kose,
2. srednji kos razpolovimo in nad njim razpnemo enakostranični trikotnik,
3. ponovimo predhodni dve točki za vsako izmed štirih dobljenih daljic.

T3

T1

T2

T4

T5

v pomoč naj vam bo naslednji kos kode:

public void drawFractal (int x1, int y1, int x5, int y5,

Graphics g)

{

int deltaX, deltaY, x2, y2, x3, y3, x4, y4;

if ( **java.lang.Math.abs(x5 - x1)<7** )

g.drawLine (x1, y1, x5, y5);

else

{

deltaX = x5 - x1;

deltaY = y5 - y1;

x2 = x1 + deltaX / 3;

y2 = y1 + deltaY / 3;

double SQ = java.lang.Math.sqrt(3.0) / 6; // višina

x3 = (int) ((x1+x5)/2 + SQ \* (y1-y5));

y3 = (int) ((y1+y5)/2 + SQ \* (x5-x1));

x4 = x1 + deltaX \* 2/3;

y4 = y1 + deltaY \* 2/3;

drawFractal (x1, y1, x2, y2, g);

drawFractal (x2, y2, x3, y3, g);

drawFractal (x3, y3, x4, y4, g);

drawFractal (x4, y4, x5, y5, g);

}

}

Sama realizacija programa naj bo taka, da bo omogočal izris fraktala določene stopnje. Pri tem naj bo stopnja 1 ravna črta, stopnja 2 kot je narisano na podani sliki, … Z mastnim tiskom v kodi je označen pogoj za končanje rekurzivnega postopka ( distanca med končnima točkama daljice po x-u pod 7 pik/pixlov). Tega bo potrebno spremeniti v npr. stopnja > 1 in dodati glavi metode dodaten parameter 'stopnja' ….

Kochovo snežinko sestavljajo trije Kochovi fraktali, povezani v enakostraničen trikotnik. Rezultat naloge je lahko tudi kochova snežinka izbrane stopnje.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Line;

import javafx.stage.Stage;

import java.util.Scanner;

public class naloga2 extends Application {

    private int stopnja = 10; // Privzeta stopnja fraktala

    public static void main(String[] args) {

        launch(args);

    }

    @Override

    public void start(Stage primaryStage) {

        Pane pane = new Pane();

        Scene scene = new Scene(pane, 800, 800);

        primaryStage.setTitle("Kochov Fraktal - Stopnja " + stopnja);

        primaryStage.setScene(scene);

        primaryStage.show();

        // Koordinati začetne in končne točke

        int x1 = 100, y1 = 400;

        int x5 = 700, y5 = 400;

        // Risanje Kochovega fraktala

        drawFractal(x1, y1, x5, y5, pane, stopnja);

    }

    public void drawFractal(int x1, int y1, int x5, int y5, Pane pane, int stopnja) {

        if (stopnja <= 1) {

            Line line = new Line(x1, y1, x5, y5);

            line.setStroke(Color.BLACK);

            pane.getChildren().add(line);

        } else {

            int deltaX = x5 - x1;

            int deltaY = y5 - y1;

            int x2 = x1 + deltaX / 3;

            int y2 = y1 + deltaY / 3;

            double SQ = Math.sqrt(3.0) / 6; // višina

            int x3 = (int) ((x1 + x5) / 2 + SQ \* (y1 - y5));

            int y3 = (int) ((y1 + y5) / 2 + SQ \* (x5 - x1));

            int x4 = x1 + deltaX \* 2 / 3;

            int y4 = y1 + deltaY \* 2 / 3;

            drawFractal(x1, y1, x2, y2, pane, stopnja - 1);

            drawFractal(x2, y2, x3, y3, pane, stopnja - 1);

            drawFractal(x3, y3, x4, y4, pane, stopnja - 1);

            drawFractal(x4, y4, x5, y5, pane, stopnja - 1);

        }

    }

}

**Naloga 3**

Cilj naloge je primerjava postopkov numerične integracije. Pri tem bomo v precep vzeli 3 postopke : aproksimacijo s pravokotniki(kvadratična metoda), aproksimacijo s trapezi(trapezoidna metoda) in Simpsonovo metodo. Poenostavitev Simpsonove metode bo podana v nadaljevanju.

Za osnovo določitev ploščin, ki jih oklepajo ovojnice funkcij z abscisno osjo bomo vzeli naslednji dve funkciji:



Določali bomo ploščine na intervalu od 0 do 5. Pri tem si dejansko ploščino prve (linearne) funkcije izračunajte sami; ploščina druge pa se izračuna, kot je podano na spodnjem primeru:



Pri kvadratični metodi ploščine posameznih pravokotnikov lahko računamo na 3 različne načine: levi, srednji, desni; odvisno od tega, katero točko vzamemo pri izračunu višine pravokotnika. V naši nalogi bomo vedno jemali srednjo, ki nam bo omogočila kasneje tudi simpsonovo aproksimacijo:

f(x)

dx

leva

srednja

desna

x

Na sliki je opaziti, da se ploščine pri kvadratični aproksimaciji razlikujejo glede na to, ali za višino pravokotnika uporabiti levo mejo intervala, desno mejo intervala ali vrednost funkcije na sredini intervala.

Simpsonova metoda

Vrednost ploščine odseka n izbrane širine (dx) se izračuna tako, da vzamete tretjino vrednosti, ki jo dobite kot rezultat izračuna ploščine istega odseka po trapezni metodi in temu prištejete dve tretjini vrednosti ploščine istega odseka izračunanega po kvadratični metodi z uporabo srednje vrednosti (midpoint). Formula:



Rezultat izvajanja programa naj bo primerjalna tabela:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Intervalov | Kvadratična(mid) | trapezna | simpsonova | izračunano |
| F1 | 1 |  |  |  |  |
|  | 2 |  |  |  |  |
|  | 4 |  |  |  |  |
|  | 10 |  |  |  |  |
|  | 50 |  |  |  |  |
|  | 100 |  |  |  |  |
| F2 | 1 |  |  |  | 11.11111 |
|  | 2 |  |  |  | 11.11111 |
|  | 4 |  |  |  | 11.11111 |
|  | 10 |  |  |  | 11.11111 |
|  | 50 |  |  |  | 11.11111 |
|  | 100 |  |  |  | 11.11111 |

import java.util.function.DoubleUnaryOperator;

public class naloga3 {

    // Funkciji iz navodil

    static double f1(double x) { // f1(x) = 1/2 x + 2

        return 0.5 \* x + 2.0;

    }

    static double f2(double x) { // f2(x) = 1/6 (x^2 + 5)

        return (x \* x + 5.0) / 6.0;

    }

    // Točna integrala na [0,5]

    static double exactF1(double a, double b) { // ∫ (1/2 x + 2) dx = 1/4 x^2 + 2x

        return 0.25 \* (b \* b - a \* a) + 2.0 \* (b - a);

    }

    static double exactF2(double a, double b) { // ∫ (x^2+5)/6 dx = 1/18 x^3 + 5/6 x

        return (Math.pow(b, 3) - Math.pow(a, 3)) / 18.0 + (5.0 / 6.0) \* (b - a);

    }

    // Kvadratnična (metoda srednjih pravokotnikov)

    static double midpoint(DoubleUnaryOperator f, double a, double b, int n) {

        double h = (b - a) / n;

        double sum = 0.0;

        for (int i = 0; i < n; i++) {

            double mid = a + (i + 0.5) \* h;

            sum += f.applyAsDouble(mid);

        }

        return sum \* h;

    }

    // Trapezna metoda

    static double trapezoid(DoubleUnaryOperator f, double a, double b, int n) {

        double h = (b - a) / n;

        double sum = 0.5 \* (f.applyAsDouble(a) + f.applyAsDouble(b));

        for (int i = 1; i < n; i++) {

            double x = a + i \* h;

            sum += f.applyAsDouble(x);

        }

        return sum \* h;

    }

    // Poenostavljena Simpsonova: S = 1/3 T + 2/3 M

    static double simpsonSimplified(DoubleUnaryOperator f, double a, double b, int n) {

        double T = trapezoid(f, a, b, n);

        double M = midpoint(f, a, b, n);

        return (T + 2.0 \* M) / 3.0;

    }

    public static void main(String[] args) {

        double a = 0.0, b = 5.0;

        int[] ns = {2, 4, 10, 50, 100};

        // Tabela za F1

        System.out.println("F1");

        System.out.println("Intervalov\tKvadratnična(mid)\ttrapezna\tsimpsonova\tizračunano");

        for (int n : ns) {

            double mid = midpoint(naloga3::f1, a, b, n);

            double tra = trapezoid(naloga3::f1, a, b, n);

            double sim = simpsonSimplified(naloga3::f1, a, b, n);

            double ex = exactF1(a, b);

            System.out.printf("%d\t\t%.6f\t\t%.6f\t%.6f\t%.6f%n", n, mid, tra, sim, ex);

        }

        System.out.println();

        // Tabela za F2

        System.out.println("F2");

        System.out.println("Intervalov\tKvadratnična(mid)\ttrapezna\tsimpsonova\tizračunano");

        for (int n : ns) {

            double mid = midpoint(naloga3::f2, a, b, n);

            double tra = trapezoid(naloga3::f2, a, b, n);

            double sim = simpsonSimplified(naloga3::f2, a, b, n);

            double ex = exactF2(a, b);

            System.out.printf("%d\t\t%.6f\t\t%.6f\t%.6f\t%.6f%n", n, mid, tra, sim, ex);

        }

    }

}

**Naloga 4**

Recimo, da imamo šahovnico (8x8) polj in šahovsko figurico skakača, s principom premikanja 1,2 in 2,1 (možni skoki skakača). Iščemo sekvenco skokov skakača, iz poljubne izbrane začetne pozicije, ki s skoki pokrije celotno šahovnico, pri tem pa posamezno polje šahovnice obišče zgolj enkrat. Spišite JavaFX varianto programa, ki bo vizualiziral pozicije skokov skakača.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Rectangle;

import javafx.scene.shape.Circle;

import javafx.scene.shape.Line;

import javafx.stage.Stage;

import javafx.scene.control.Label;

import javafx.geometry.Pos;

import javafx.animation.Timeline;

import javafx.animation.KeyFrame;

import javafx.util.Duration;

public class naloga4 extends Application {

    private static final int SIZE = 8; // Velikost šahovnice

    private static final int TILE\_SIZE = 60; // Velikost posameznega polja

    private int[][] board = new int[SIZE][SIZE]; // Šahovnica (shrani številko poteze 1..64)

    private final int[] moveX = {1, 2, 2, 1, -1, -2, -2, -1}; // Možni premiki skakača v X smeri

    private final int[] moveY = {2, 1, -1, -2, -2, -1, 1, 2}; // Možni premiki skakača v Y smeri

    private Pane tilesPane;   // plast za polja

    private Pane overlayPane; // plast za poti, številke in skakača

    private Timeline timeline; // animacija

    public static void main(String[] args) {

        launch(args);

    }

    @Override

    public void start(Stage primaryStage) {

        Pane root = new Pane();

        tilesPane = new Pane();

        overlayPane = new Pane();

        root.getChildren().addAll(tilesPane, overlayPane);

        Scene scene = new Scene(root, SIZE \* TILE\_SIZE, SIZE \* TILE\_SIZE);

        primaryStage.setTitle("Skakač na šahovnici – klikni začetno polje");

        primaryStage.setScene(scene);

        primaryStage.show();

        drawEmptyBoard();

        // Klik na šahovnico določi začetno polje in sproži vizualizacijo

        root.setOnMouseClicked(e -> {

            int x = (int) (e.getX() / TILE\_SIZE);

            int y = (int) (e.getY() / TILE\_SIZE);

            if (x < 0 || x >= SIZE || y < 0 || y >= SIZE) return;

            // prekini morebitno prejšnjo animacijo in očisti prekrivno plast

            if (timeline != null) {

                timeline.stop();

            }

            overlayPane.getChildren().clear();

            // izračun poti (Warnsdorff)

            boolean ok = computeTourWarnsdorff(x, y);

            if (!ok) {

                System.out.println("Ni rešitve iz te začetne pozicije.");

                return;

            }

            // animiraj pot (številke na poljih + premik skakača + povezovalne črte)

            animateTour();

        });

    }

    // Warnsdorffov algoritem: hitro poišče turnejo iz poljubnega začetnega polja

    private boolean computeTourWarnsdorff(int startX, int startY) {

        // počisti šahovnico

        for (int i = 0; i < SIZE; i++) {

            for (int j = 0; j < SIZE; j++) {

                board[i][j] = 0;

            }

        }

        int x = startX;

        int y = startY;

        board[x][y] = 1;

        for (int move = 2; move <= SIZE \* SIZE; move++) {

            int bestNX = -1, bestNY = -1;

            int bestDegree = Integer.MAX\_VALUE;

            for (int k = 0; k < 8; k++) {

                int nx = x + moveX[k];

                int ny = y + moveY[k];

                if (isSafe(nx, ny)) {

                    int deg = countOnwardMoves(nx, ny);

                    if (deg < bestDegree) {

                        bestDegree = deg;

                        bestNX = nx;

                        bestNY = ny;

                    }

                }

            }

            if (bestNX == -1) {

                // zatajilo – ni naslednjega koraka

                return false;

            }

            x = bestNX;

            y = bestNY;

            board[x][y] = move;

        }

        return true;

    }

    // Preveri, če je naslednji premik varen

    private boolean isSafe(int x, int y) {

        return x >= 0 && x < SIZE && y >= 0 && y < SIZE && board[x][y] == 0;

    }

    // Prešteje koliko varnih izhodov ima polje (nx, ny)

    private int countOnwardMoves(int nx, int ny) {

        int count = 0;

        for (int k = 0; k < 8; k++) {

            int tx = nx + moveX[k];

            int ty = ny + moveY[k];

            if (isSafe(tx, ty)) count++;

        }

        return count;

    }

    // Nariše (samo) polja šahovnice – brez številk in poti

    private void drawEmptyBoard() {

        tilesPane.getChildren().clear();

        for (int i = 0; i < SIZE; i++) {

            for (int j = 0; j < SIZE; j++) {

                Rectangle tile = new Rectangle(i \* TILE\_SIZE, j \* TILE\_SIZE, TILE\_SIZE, TILE\_SIZE);

                tile.setFill((i + j) % 2 == 0 ? Color.BEIGE : Color.SADDLEBROWN);

                tilesPane.getChildren().add(tile);

            }

        }

    }

    // Animira pot: na vsako polje doda številko poteze, črto med potezama in premika krog (skakača)

    private void animateTour() {

        // Poišči zaporedje koordinat iz matrike board (1..N)

        int total = SIZE \* SIZE;

        int[] xs = new int[total];

        int[] ys = new int[total];

        for (int step = 1; step <= total; step++) {

            outer:

            for (int i = 0; i < SIZE; i++) {

                for (int j = 0; j < SIZE; j++) {

                    if (board[i][j] == step) {

                        xs[step - 1] = i;

                        ys[step - 1] = j;

                        break outer;

                    }

                }

            }

        }

        overlayPane.getChildren().clear();

        Circle knight = new Circle(TILE\_SIZE / 4.0);

        knight.setFill(Color.RED);

        overlayPane.getChildren().add(knight);

        timeline = new Timeline();

        timeline.setCycleCount(Timeline.INDEFINITE);

        for (int idx = 0; idx < total; idx++) {

            final int i = idx; // za lambda

            Duration t = Duration.millis(200 \* i);

            timeline.getKeyFrames().add(new KeyFrame(t, ev -> {

                int cx = xs[i];

                int cy = ys[i];

                // premik skakača

                knight.setCenterX(cx \* TILE\_SIZE + TILE\_SIZE / 2.0);

                knight.setCenterY(cy \* TILE\_SIZE + TILE\_SIZE / 2.0);

                // številka poteze na polju

                Label lbl = new Label(String.valueOf(i + 1));

                lbl.setMinSize(TILE\_SIZE, TILE\_SIZE);

                lbl.setAlignment(Pos.CENTER);

                lbl.setTextFill(((cx + cy) % 2 == 0) ? Color.BLACK : Color.WHITE);

                lbl.setLayoutX(cx \* TILE\_SIZE);

                lbl.setLayoutY(cy \* TILE\_SIZE);

                overlayPane.getChildren().add(lbl);

                // črta od prejšnjega polja do trenutnega

                if (i > 0) {

                    Line line = new Line(

                        xs[i - 1] \* TILE\_SIZE + TILE\_SIZE / 2.0,

                        ys[i - 1] \* TILE\_SIZE + TILE\_SIZE / 2.0,

                        cx \* TILE\_SIZE + TILE\_SIZE / 2.0,

                        cy \* TILE\_SIZE + TILE\_SIZE / 2.0

                    );

                    line.setStroke(Color.DARKRED);

                    line.setStrokeWidth(2);

                    overlayPane.getChildren().add(line);

                }

            }));

        }

        // Ustavi animacijo po zadnjem koraku

        timeline.getKeyFrames().add(new KeyFrame(Duration.millis(200 \* total + 10), ev -> timeline.stop()));

        timeline.playFromStart();

    }

    // (Neobvezno) izpiše zaporedje obiskanih polj v terminal (po korakih 1..N)

    private void printVisited() {

        System.out.println("Zaporedje obiskanih polj (korak: (x, y)):");

        for (int step = 1; step <= SIZE \* SIZE; step++) {

            boolean found = false;

            for (int x = 0; x < SIZE && !found; x++) {

                for (int y = 0; y < SIZE && !found; y++) {

                    if (board[x][y] == step) {

                        System.out.printf("%3d: (%d, %d)%n", step, x, y);

                        found = true;

                    }

                }

            }

        }

    }

}

**Naloga 5**

Predpostavite, da imate vertikalno črto (pravokotno na osnovnico koordinatnega sistema). Na tretjini višine od osnovnice požene (hm) drevo novo vejo v obliki daljice, od osnovnice proti najvišji točki višine drevesa, vendar je nagnjena od osnovnice v levo za npr. 50. Vsaka veja se deli na isti način. Delitve ni več, ko je veja krajša od 5 pik. Dve iteraciji sta dani na spodnji sliki. Sestavite ustrezen JavaFX program, ki bo 'drevo' vizualiziral.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Line;

import javafx.stage.Stage;

import java.util.Scanner;

public class naloga5 extends Application {

    private double angle = Math.toRadians(30); // Kot nagnjenja vej

    private double lengthFactor = 0.67; // Faktor skrajšanja vej

    private int minLength = 5; // Minimalna dolžina veje za nadaljevanje delitve

    public static void main(String[] args) {

        launch(args);

    }

    @Override

    public void start(Stage primaryStage) {

        Pane pane = new Pane();

        Scene scene = new Scene(pane, 800, 600);

        primaryStage.setTitle("Fraktalno Drevo");

        primaryStage.setScene(scene);

        primaryStage.show();

        // Začetna točka in dolžina glavne veje

        double startX = scene.getWidth() / 2;

        double startY = scene.getHeight();

        double length = scene.getHeight() / 3;

        // Nariši drevo

        drawTree(pane, startX, startY, length, -Math.PI / 2); // -Math.PI/2 za vertikalno gor

    }

    private void drawTree(Pane pane, double x1, double y1, double length, double angle) {

        if (length < minLength) return; // Pogoji za končanje rekurzije

        // Izračun končne točke veje

        double x2 = x1 + length \* Math.cos(angle);

        double y2 = y1 + length \* Math.sin(angle);

        // Nariši vejo

        Line line = new Line(x1, y1, x2, y2);

        line.setStroke(Color.BROWN);

        pane.getChildren().add(line);

        // Rekurzivno nariši levi in desni del veje

        drawTree(pane, x2, y2, length \* lengthFactor, angle + this.angle); // Levi del

        drawTree(pane, x2, y2, length \* lengthFactor, angle - this.angle); // Desni del

    }

}

**Naloga 6**

Sestavite javanski program, ki bo izrisoval trikotnik Sierpinskega. Stopnjo (št. korakov) izrisovanja omejite tako, da prenehate izrisovati, ko dolžina najkrajše stranice trikotnika pade pod 5 pik. Dela nimate veliko, zgolj preverite, popravite in izboljšajte predstavljen postopek.

Kot opomba :

Postope izvedete v fazah kot:

1. vsako stranico trikotnika razpolovimo,
2. izrišemo črte med razpolovišči stranic
3. ponovimo postopek za vsakega od nastalih trikotnikov (ali za vse, brez notranjega)

predlagam metodo : sierpinski(tocka1, tocka2,tocka3); // tockaX predstavlja oglišče trikotnika

mimogrede:

Tn3

T3

T2

T1

Tn1

Tn2

Iz trikotnika (T1,T2,T3) na vsakem koraku nastanejo štirje trikotniki:

* (T1,Tn1,Tn3),
* (Tn1,Tn2,Tn3),
* (Tn1,T2,Tn2),
* (Tn3,Tn2,T3)

import javafx.application.Application;

import javafx.geometry.Point2D;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Line;

import javafx.stage.Stage;

public class naloga6 extends Application {

    private static final double MIN\_SIDE = 5.0; // prag za ustavitev rekurzije

    public static void main(String[] args) {

        launch(args);

    }

    @Override

    public void start(Stage stage) {

        Pane pane = new Pane();

        Scene scene = new Scene(pane, 800, 700);

        stage.setTitle("Sierpinski trikotnik");

        stage.setScene(scene);

        stage.show();

        // Začetni (približno) enakostranični trikotnik v oknu

        double margin = 40;

        double side = scene.getWidth() - 2 \* margin;

        double height = side \* Math.sqrt(3) / 2.0;

        double cx = scene.getWidth() / 2.0;

        double baseY = margin + height; // osnovnica spodaj

        Point2D T1 = new Point2D(cx - side / 2.0, baseY);      // levo-spodaj

        Point2D T2 = new Point2D(cx + side / 2.0, baseY);      // desno-spodaj

        Point2D T3 = new Point2D(cx, baseY - height);          // zgoraj

        // (neobvezno) narišemo obris začetnega trikotnika

        drawEdgeTriangle(pane, T1, T2, T3, Color.BLACK, 1.5);

        // Sierpinski rekurzija

        sierpinski(pane, T1, T2, T3);

    }

    // Rekurzivni izris Sierpinskega trikotnika (prek črt med razpolovišči)

    private void sierpinski(Pane pane, Point2D a, Point2D b, Point2D c) {

        double s1 = a.distance(b);

        double s2 = b.distance(c);

        double s3 = c.distance(a);

        double min = Math.min(s1, Math.min(s2, s3));

        if (min < MIN\_SIDE) return; // baza rekurzije

        // Razpolovišča stranic

        Point2D ab = a.midpoint(b);

        Point2D bc = b.midpoint(c);

        Point2D ca = c.midpoint(a);

        // Nariši notranji trikotnik, ki povezuje razpolovišča

        drawEdgeTriangle(pane, ab, bc, ca, Color.DARKBLUE, 1.0);

        // Rekurzija na treh zunanjih trikotnikih (brez notranjega)

        sierpinski(pane, a, ab, ca);

        sierpinski(pane, ab, b, bc);

        sierpinski(pane, ca, bc, c);

    }

    // Pomožna: nariše obris trikotnika podanega s tremi točkami

    private void drawEdgeTriangle(Pane pane, Point2D p1, Point2D p2, Point2D p3, Color color, double width) {

        pane.getChildren().add(makeLine(p1, p2, color, width));

        pane.getChildren().add(makeLine(p2, p3, color, width));

        pane.getChildren().add(makeLine(p3, p1, color, width));

    }

    private Line makeLine(Point2D p, Point2D q, Color color, double width) {

        Line l = new Line(p.getX(), p.getY(), q.getX(), q.getY());

        l.setStroke(color);

        l.setStrokeWidth(width);

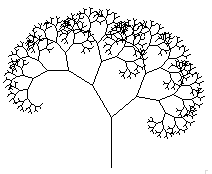
        return l;

    }

}

**Naloga 7**

Naloga je variacija naloge 6. Pri tem je cilj izrisati drevo, kot je podano na spodnji sliki (fraktal, ki izgleda kot drevo). Če ga pogledate malo bolje, vidite, da je drevo sestavljeno iz debla in dveh vej. Vsak izmed vej je dejansko 'enako' drevo, le manjše, pa malo nagnjeno. V bistvu pri implementaciji rišemo veje (črte), torej bomo za izris verjetno spisali metod s tremi parametri : Točko, ki predstavlja prijemališče oz. koordinato veje/debla, dolžino debla in kot, pod katerim se izriše deblo:

npr.:

narisiDrevo(Tocka izhodisceDebla, double dolzinaDebla, double kot);

pri tem je Tocka {double x, double y}

v vsaki iteraciji rišemo levo in desno poddrevo in sicer tako, da kot leve veje vsakič spremenimo za npr. 30o, desno pa za 50o.

Dolžina leve veje naj bo npr. 75%dolžine debla, desna pa 66% dolžine debla.

Npr.: drevo rišemo v sekvenci (pseudo):

narisiDrevo(t1, dolzinaDebla \* 0.75, kot + 30stopinj);  
narisiDrevo(t1, dolzinaDebla \* 0.66, kot – 50stopinj);

Kreirajte drevo po danih zahtevah

Skušajte kreirati drugačno drevo s spremembo dolžin in kotov

Skušajte narediti drevo nepravilno (s tem, da določene veje ne izrišete, npr.: vejo izrišete z verjetnostjo 1/6.

**Naloga 8**

Variirajte nalogo 6, za osnovo vzemite pravokotnik (štirikotnik).

1. Korak rekurzije razpolovi stranice pravokotnika in v razpolovišča vpne štirikotnika. Ponavljajte vpenanje, dokler krajša stranica dolžinsko ne pade pod 10 enot (pik).
2. Korak rekurzije razpolovi stranice pravokotnika in poveže nasproti ležeča razpolovišča – razdeli pravokotnik v 4 pravokotnike. Nato naključno izbere enega od dobljeni in na njem ponovi postopek razpolavljanja. Postopek končamo, ko je ena od dobljenih strani krajša od 10 pik.