02101 Indledende Programmering
Hjemmeopgave 2

s203775 s194549 s204747

29. oktober, 2021

Arbejdsfordeling

Vi samarbejdede om alle opgaver, men uddelte ansvarsområder til forskellige gruppemedlemmer.

s203775 var ansvarlig for problem 3.
s194549 var ansvarlig for problem 2.
s204747 var ansvarlig for problem 1.

1. Problem 1

2. Problem 2: Triangular Patterns

Problemet er blevet løst ved at lave en constructor som laver et to-dimensionelt array på størrelse med de værdier der er blevet givet. Så laver den den første række baseret på initial, og ud fra dette kan alle andre værdier i det 2-dimensionelle array blive bestemt via. definitionerne givet i opgavebeskrivelsen, altså et triangulært mønster.

3. Problem 3: Polar Points

I problemet designes en klasse `MovingPoint` der beskriver et punkt-objekt der kan bevæge sig efter polære koordinater (retning og længde), og desuden kan instantieres arbitrært mange gange.

Da ethvert punk er beskrevet i et 2-dimensionelt vektorrum, defineres også en underklasse `Vector`, som gør det muligt at behandle en `double[]` array som en matematisk vektor, med tilhørende simple operationer: addition og mulitplikation med skalar. Givet disse operationer er det nemlig muligt at "bevæge" punktet rundt (addition), i en givet *retning* og *hastighed* (skalleret retningsvektor med multiplikation).

Klassen `MovingPoint` har altså 3 egenskaber på ethvert tidspunkt, netop en position, retning og hastighed - på klassen givet ved `this.position` (`Vector`), `this.direction` (`double`) og `this.speed` (`double`).

Det bemærkes at der i opgaven er tale om et uendeligt stort vektorrum - altså ingen begrænsning af et punkts position. Til gengæld er egenskaberne begrænset i både retningen (vinklen) [grader°] θ, som er angivet i værdierne $0 \leq \theta < 360$, samt hastigheden som afskæres i intervallet $[0, 20]$.

Fremfor at tage højde for dette alle steder i koden hvor hastighed eller vinkel bliver sat, udstilles istedet 2 instans metoder til at sætte værdierne "sikkert", `setDirection` og `setSpeed`. Disse metoder indeholder implementeringen til at begrænse værdierne. Hastigheden er simpel, og skal blot skæres af i intervallets grænser, hvis den bliver sat med værdier udenfor intervallet. Da vinklen er cyklisk, skal den ikke blot skæres af i grænserne, men istedet "dreje rundt til næste gyldige vinkel". Denne funktionalitet opnås ved at gøre brug af `%` (modulus) til `360`. Dette fjerner dog ikke muligheden for negative værdier, hvorfor man blot kan trække en negativ vinkel fra 360 når dette er tilfældet:

---

if θ < 0 **then return** 360 - (θ % 360)
else return θ % 360
end if

Med disse *setter* funktioner er det muligt at ignorere værdiernes begrænsninger i det øvrige logiske flow af koden. Årsagen til denne tilgang fremfor en direkte implementering i eksempelvis `accelerateBy()` eller `turnBy()` er, at værdierne også bliver sat i *andre* metoder. På den valgte måde kan enhver instansmetode eller constructor, som eksistere i koden nu (eller kommer til det i fremtiden), frit sætte værdierne uden at disse bliver sat udenfor deres angivede begrænsninger.

Dette valg gør at den øvrige implementering var let at overskue, og nærmest bare skulle aflæses fra opgavebeskrivelsen. `move` metoden er interessant da den implementere utility funktionerne fra `Vector` typen.

I `move()` metoden konverteres `this.direction` fra grader til radianer (`Math.toRadians`), hvorefter en retningsvektor kan udledes af vinklen vha. `Math.cos()` og `Math.sin()`. Denne retningsvektor skalleres så med hastigheden mulitpliceret over `duration`, som altså giver den resulterende ændring i position af bevægelsen. Til sidst kan ændrings-vektoren derfor lægges til `this.position`, hvorved altså den endelige resulterende position er fundet.

Bemærk at `Vector` klassen både udstiller de matematiske operationer som statiske metoder og som instansmetoder. I denne implementering gøres kun brug af instansmetoderne, men de statiske er stadigvæk udstillet i tilfælde af at koden skal udvides, hvorved det i nogle tilfælde kunne give mere mening at bruge

de statiske metoder som har en mere explicit syntax, for at forbedre læsbarheden. Instansmetoderne er afledt af de statiske metoder, hvorfor de stadigvæk er inkluderet i koden.

Se Bilag: Java klasser.

4. Bilag: Java klasser

Problem 1

Problem 2

```java
import java.util.Arrays;

public class TrianglePattern {
  private int h, n;
  private int[] initial;
  private int[][] grid;

  public static void main(String[] args) {
    int n = 44;
    int h = 20;
    int[] init = new int[] { 30, 2, 17 };
    TrianglePattern tp = new TrianglePattern(n, h, init);
    System.out.println(n);
    System.out.println(h);
    System.out.println(Arrays.toString(init));

    System.out.println(tp.getN());
    System.out.println(tp.getH());
    System.out.println(Arrays.toString(tp.getInitial()));

    for (int r = 0; r < tp.getH(); r++) {
      for (int c = 0; c < tp.getN(); c++) {
        System.out.print(tp.getValueAt(r, c));
      }
      System.out.println("");
    }

  }

  public TrianglePattern(int n, int h, int[] initial) {
    // Først initaliserer den et 2d array med alle værdier gemt i
    int[][] grid = new int[h][n];
    // Så laver den den første række baseret på initial
    for (int c = 0; c < grid[0].length; c++)
      for (int j = 0; j < initial.length; j++) {
        if (c == initial[j]) {
          grid[0][c] = 1;
        } else {
          // Her sørger den for ikke at lave dubletter eller overskride
             forrige værdier i
          // initial
          if (grid[0][c] == 1 || grid[0][c] == 0) {
            continue;
          } else {
            grid[0][c] = 0;
          }
        }
```

```
47        }
48      // Så fra række 2 går den igennem alle colonner og rækker, og giver
            dem værdier
49      // i griddet
50      for (int r = 1; r < h; r++) {
51        for (int c = 0; c < n; c++) {
52          // Den første og sidste værdi vil altid være 0
53          if (c == 0 || c == n - 1) {
54            grid[r][c] = 0;
55          } else {
56            // Ellers er det lavet på et trekantsmønster
57            if (grid[r - 1][c - 1] == 0) {
58              if (grid[r - 1][c] == 0 && grid[r - 1][c + 1] == 0) {
59                grid[r][c] = 0;
60              } else {
61                grid[r][c] = 1;
62              }
63            } else {
64              if (grid[r - 1][c] == 0 && grid[r - 1][c + 1] == 0) {
65                grid[r][c] = 1;
66              } else {
67                grid[r][c] = 0;
68              }
69            }
70          }
71        }
72      }
73      // Til sidst gør jeg så man kan kalde alle variable
74      this.n = n;
75      this.h = h;
76      this.initial = initial;
77      this.grid = grid;
78    }
79
80    // Laver funktionerder kan give de overordnede værdier for griddet,
          samt de
81    // individuelle værdier inde i den.
82    public int getN() {
83      int column = n;
84      return column;
85    }
86
87    public int getH() {
88      int row = h;
89      return row;
90    }
91
92    public int[] getInitial() {
93      int[] init;
94      init = initial;
95      return init;
96    }
97
98    public int getValueAt(int r, int c) {
99      int value;
100     value = grid[r][c];
```

```
    return value;
  }

}
```

Problem 3

```
public class MovingPoint {
  Vector position;
  double direction;
  double speed;

  public void setDirection(double d) {
    double wrapped = d % 360;
    double clamped = clamp(wrapped, 0, 360);
    this.direction = wrapped < 0 ? 360 + wrapped : clamped;
  }

  public void setSpeed(double s) {
    this.speed = clamp(s, 0, 20);
  }

  public MovingPoint() {
    this.position = new Vector(2);
    this.direction = 90;
    this.speed = 0;
  }

  public MovingPoint(double x, double y, double direction, double speed)
      {
    this.position = new Vector(new double[] { x, y });
    this.setDirection(direction);
    this.setSpeed(speed);
  }

  public void move(double duration) {
    double rad = Math.toRadians(this.direction);
    double[] coords = new double[] { Math.cos(rad), Math.sin(rad) };
    Vector dir = new Vector(coords);
    this.position = this.position.add(dir.scale(this.speed * duration));
  }

  public void accelerateBy(double change) {
    this.setSpeed(this.speed + change);
  }

  public void turnBy(double angle) {
    this.setDirection(this.direction + angle);
  }

  public String toString() {
    return this.position.toString() + " " + this.direction + " " + this.
        speed;
  }
```

```
46
47    static double clamp(double input, int min, int max) {
48      return input > max ? max : input < min ? min : input;
49    }
50
51    static class Vector {
52      private final double[] array;
53
54      public Vector(int d) {
55        this.array = new double[d];
56      }
57
58      public Vector(double[] a) {
59        this.array = a;
60      }
61
62      public double get(int index) {
63        return this.array[index];
64      }
65
66      public int dimension() {
67        return this.array.length;
68      }
69
70      // overwrite toString() method to print formatted vector
71      // instead of primitive / native array.toString()
72      public String toString() {
73        String[] collect = new String[this.array.length];
74        for (int i = 0; i < this.array.length; i++) {
75          collect[i] = String.valueOf(this.array[i]);
76        }
77        return "[" + String.join(";", collect) + "]";
78      }
79
80      public Vector add(Vector V) {
81        return addition(new Vector(this.array), V);
82      }
83
84      public Vector scale(double s) {
85        return scale(new Vector(this.array), s);
86      }
87
88      public static Vector addition(Vector A, Vector B) {
89        double[] result = new double[A.dimension()];
90        for (int i = 0; i < result.length; i++) {
91          result[i] = (A.get(i) + B.get(i));
92        }
93        return new Vector(result);
94      }
95
96      public static Vector scale(Vector V, double s) {
97        double[] result = new double[V.dimension()];
98        for (int i = 0; i < result.length; i++) {
99          result[i] = V.get(i) * s;
100       }
101       return new Vector(result);
```

```
102         }
103     }
104 }
```