2025 Fall Semester

Gruppeaflevering-2

02100

Indledende Programmering og Softwareteknologi, Efterår 2025

Student: Sebastian Faber Steffensen (s255609), Mikkel M.H. Pedersen (s255015) , Rasmus Rosendahl-Kaa (s255955)

Due: 5. Oktober 2025

Programming Language: Java

Arbejdsdeling

Sebastian: Hovedansvar for BuffonsNeedle

Mikkel: Hovedansvar for Palindrome programmet og RomanNumerals.

Rasmus: Hovedansvar for RandomWalk

Vi har samarbejdet om alle opgaver og diskuteret designvalg sammen.

Overordnet Programbeskrivelse

Afleveringen består af fire Java-programmer:

- **RomanNumerals**: Konverterer fra positive heltal til romer tal
- **Palindrome**: Checker om en tekst er et palindrom (ignorerer ikke-bogstaver)
- **BuffonsNeedle**: Monte Carlo simulation til at estimere værdien af π ved hjælp af Buffon's nåleproblem
- **RandomWalk**: Simulerer tilfældig gåtur i et gitter med grafisk visualisering

Alle programmer bruger korrekt navngivning og indrykning som undervist på kurset.

Problem 1

Skriv et program `RomanNumerals` som konverterer et heltal i intervallet $1-3999$ til romertal. Det vil sige, at hvis f.eks. inputtet er `6` skal outputtet være `VI` og hvis inputtet er 29 skal outputtet være `XXIX`. Se om reglerne for dannelse af romertal på nettet (søg på "roman numerals").

Specification

Programmet skal:

- Spørge om, hvad tal skal konverteres til romersk fra positive heltal mellem $1 - 3999$.
- Konvertere positive heltal ($1 - 3999$) til romerske
- Håndtere alle gældende romerske talregler

`Scanner` bliver brugt til, at læse bruger input. Vi bruger to arrays, et som holder tal i ti-potenser og halv-ti potenser: $1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900$ og 1000. Ligeledes et array som holder de tilsvarende romer tal: `I, IV, V, IX, X, XL, L, XC, C, CD, D, CM` og `M`. Vi iterere over vores input tal, og hvis det er større end det givne ti potens subtrahere vi det og lægger det tilsvarende romer tal til en streng, som til sidst bliver returneret.

Test Case

- Test af RomanNumerals med forskellige konverteringer

```
❯ java RomanNumerals.java
Enter positive number to convert: 3450
MMMCDL


❯ java RomanNumerals.java
Enter positive number to convert: 550
DL

❯ java RomanNumerals.java
Enter positive number to convert: 2697
MMDCXCVII
```

Kommentar: Programmet håndterer korrekt komplekse tal som 1999 (MCMXCIX) og kan konvertere begge veje.

Problem 2

Skriv et program `Palindrome` som checker om en tekst er et palindrom. Det vil sige om det læses ens forfra og bagfra som f.eks. “Ryd vores dug. Ret dog netgardinet. Tak, Egil. Viggo, gid da denne dame så de samer af dem, der red med fare, mase dåsemaden ned ad dig. Og giv lige katten i dragten godter. Gud se, rovdyr!”(i Guinness’ Rekordbog 1985 ifølge http://www.palindromer.dk). Du kan læse om reglerne for palindromer på nettet (søg på “palin- drome”). Bemærk at der ikke tages hensyn til forskellen mellem store og små bogstaver, og at der ses bort fra alle tegn som ikke er bogstaver

Specification

Programmet skal:
- Tage en tekstlinje som input
- Ignorere alle tegn der ikke er bogstaver
- Sammenligne teksten med sig selv bagvendt
- Outputte om teksten er et palindrom eller ej

`Scanner` blev brugt til, at tage bruger input. Vi stod mellem to måder at implementere dette program, det første brugte vi `for` loops til at først rengøre input teksten, og fjerne ikke-bogstaver og mellemrum, derefter kan vi igen iterere igennem vores rengjort tekst, og se om karakteren er det samme i `charAt(i) == charAt(text.length - 1 - i)`. Vi valgte dog at gøre det med regex i stedet, da denne implementation er meget kortere. Vi bruger regex’en `[^a-zA-Z]` som fjerner alle ikke bogstaver, dernæst konverteret vi til `lowercase`, og lavede en `equals`, med den omvendte version af strengen.

Test Case
- Test af Palindrome med forskellige input

```
❯ java Palindrome.java
Enter line to check: A man, a plan, a canal: Panama
"A man, a plan, a canal: Panama" is a palindrome!

❯ java Palindrome.java
Enter line to check: Hello World
"Hello World" is not a palindrome!

❯ java Palindrome.java
Enter line to check: racecar
"racecar" is a palindrome!

❯ java Palindrome.java
Enter line to check: Ryd vores dug. Ret dog netgardinet. Tak, Egil.
Viggo, gid da denne dame så de samer af dem, der red med fare, mase
dåsemaden ned ad dig. Og giv lige katten i dragten godter. Gud se,
rovdyr!
"Ryd vores dug. Ret dog netgardinet. Tak, Egil. Viggo, gid da denne
dame så de samer af dem, der red med fare, mase dåsemaden ned ad dig.
Og giv lige katten i dragten godter. Gud se, rovdyr!" is a palindrome!
```

Kommentar: Programmet håndterer korrekt både simple palindromer og komplekse sætninger med tegnsætning og mellemrum.

Problem 3

Implementér Buffon's nåleproblem til at estimere π

Specification

Programmet skal:
- Tage antal iterationer som input
- Simulere nåle der kastes på parallelle linjer
- Beregne π baseret på hvor mange nåle krydser linjerne
- Outputte resultat i format: $\text{iterationer} \ / \ \text{krydsninger} = \pi\text{-estimat}$

Vi bruger Monte Carlo simulation hvor nålen har længde 1 og afstanden mellem linjer er 2. For hver nål genereres tilfældige værdier for position og vinkel, og vi tjekker om den krydser en linje ved at sammenligne den vertikale udstrækning med afstanden til nærmeste linje.

Test Case
- Test af BuffonsNeedle med forskellige, stigende, antal iterationer.

```
$ java BuffonsNeedle.java
Enter number of iterations: 1000
1000 / 314 = 3.1847133757961785 (2 korrekte cifrer)

$ java BuffonsNeedle.java
Enter number of iterations: 1000000
1000000 / 317759 = 3.147039108254998 (3 korrekte cifrer)

$ java BuffonsNeedle.java
Enter number of iterations: 10000000
10000000 / 3182969 = 3.1417208273156287 (4 korrekte cifrer)
```

Kommentar: Resultatet er tæt på $\pi \approx 3.14159$. Jo flere iterationer, jo mere præcist bliver estimatet. Dog bør man overveje Law of diminishing returns, da det tog markant længere tid jo flere iterationer man valgte.

Problem 4

Simulér en tilfældig gåtur (random walk) med grafisk visualisering

Specification

Programmet skal:
- Tage gitterstørrelse som input
- Starte fra centrum (0,0)
- Bevæge sig tilfældigt i 4 retninger
- Tegne punkter for hver ny position grafisk med StdDraw
- Stoppe når punktet forlader gitteret
- Udskrive antal trin og position efter hvert trin

Vi bruger `Scanner` til at få input fra brugeren og vi bruger `Random.nextint(4)` til at vælge hvilken retning, som punktet skal bevæge sig i (0 = op, 1 = ned, 2 = til venstre og 3 = til højre). Vi kører det i et while-loop, som tjekker om punktets koordinater er inde for den gridSize, som brugeren har indtastet. Vi printer også punktets position og tegner et punkt med `StdDraw` og opdaterer hvor mange iterationer der er gået, som vi printer til sidst, når while-loopet er færdigt.

Vi tjekker også om inputtet fra brugeren er en integer, og giver en fejl, hvis det ikke er det. Vi har valgt at acceptere, at brugeren indtaster et negativt tal, ved bare at se bort fra det ved at tage absolut-værdien af brugerens indtastede tal.

Størrelsen på punkterne, der tegnes, er sat til `2 / (gridSize * 4)` da vi fandt det gav en fin punktstørrelse ved forskellige gridSize.

Kommentar: Vores aflevering inkluderer ikke `StdDraw`, som vi bruger til at tegne figurene, da vi har forstået på opgaveformuleringen, at vi ikke skal afleverer filer, vi har fået udleveret. Vi havde også problemer med at kører programmet med `StdDraw` i diverse IDE's, men vi fik det til at virke i VSCode.

Test Case

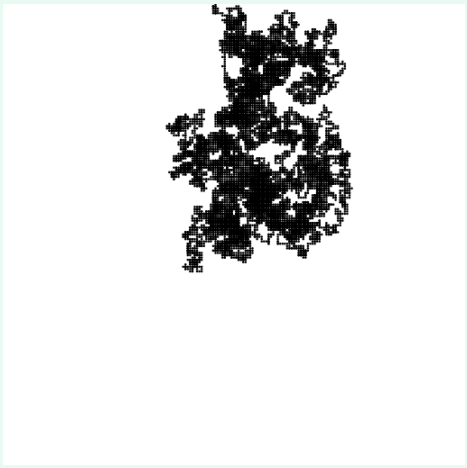
- Test af RandomWalk med gridSize 0, 100, 1000, -100, 50.0 og "hej". Program burde acceptere alle undtagen de to sidste, hvor den burde give fejlen "Not an integer". Figurene ses under testene.

```
$ java RandomWalk
Enter gridsize: 0
Position: (0, -1)
Total number of steps: 1

$ java RandomWalk
Enter gridsize: 100
...
Position: (-9, 99)
Position: (-8, 99)
Position: (-8, 100)
Position: (-8, 101)
Total number of steps: 18797

$ java RandomWalk
Enter gridsize: 1000
...
Position: (-999, 318)
Position: (-999, 317)
Position: (-1000, 317)
Position: (-1001, 317)
Total number of steps: 1312878

$ java RandomWalk
Enter gridsize: -100
...
Position: (-99, -30)
Position: (-99, -29)
Position: (-100, -29)
Position: (-101, -29)
Total number of steps: 11998
```

```
$ java RandomWalk
Enter gridsize: 50.0
Exception in thread "main" java.lang.IllegalArgumentException: Not a
number
        at RandomWalk.main(RandomWalk.java:12)

$ java RandomWalk
Enter gridsize: hej
Exception in thread "main" java.lang.IllegalArgumentException: Not a
number
        at RandomWalk.main(RandomWalk.java:12)
```

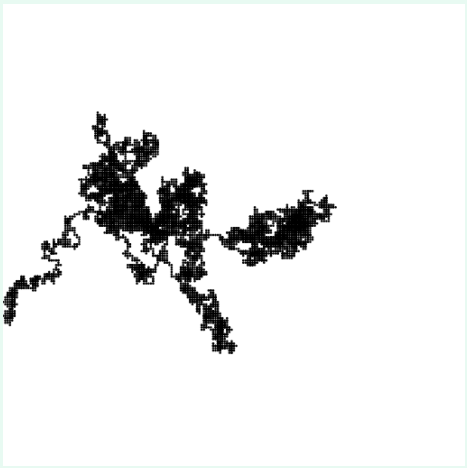
Kommentar: Programmet viser hver position og stoppes når punktet når grænsen (f.eks. −1001 er udenfor [−1000,1000]) Programmet håndterede også inputtet `0` korrekt ved at stoppe efter første iteration, hvor punktet så er udenfor gitteret. Derudover håndterede det et negativt input som forventet, og gav fejl, når inputtet ikke var en integer.

Figur 1: Output af `RandomWalk` med `gridSize` 0. Er helt tomt, som det burde være.



Figur 2: Output af `RandomWalk` med `gridSize` 100.

Figur 3: Output af `RandomWalk` med `gridSize` 1000.



Figur 4: Output af `RandomWalk` med `gridSize` −100.

Konklusion

Alle fire programmer virker korrekt og demonstrerer forskellige programmeringskoncepter: Monte Carlo simulation, string-håndtering, grafisk programmering og algoritmer til talkonvertering. Programmerne følger kursets retningslinjer for kodestil og er velstrukturerede med passende kommentarer. Endvidere blev et markup sprog brugt til rapport skrivning, ikke LaTeX, men typst, hvilket er et LaTeX alternativt.