Rapport for Anden Aflevering I Kurset intro til Programmering

Skrevet af Mads Frimann Madsen (S134109) og Magnus Andersen Vogel (S132885)

Arbejdsfordeling

Koden blev skrevet parallelt af begge hvorefter de forskellige opgaver blev sammenlignet og den bedste version af hver opgave blev herefter fundet. Det er Magnus Andersen Vogels opgave 1 vi bruge mens det er Mads Frimann Madsens opgave 2 og 3. Rapportens afsnit 1 er skrevet af Magnus Andersen Vogel, mens afsnit 2 er skrevet i fællesskab og afsnit 3 er skrevet af Mads Frimann Madsen.

Indholdsfortegnelse

Opgave 1 - Romertal

I denne opgave vil vi lave et program som kan tage et heltal mellem 0 og 3999, begge tal inkluderet, og "oversætte" det til romertal. Vi vil gøre det ved hjælp af for løkker og integer division.

Programmet består af 2 metoder, RomanNumerals og adding.

RomanNumerals får fra en scanner en integer. Denne bruger den som argument og opdeler det i forskellige tal. Først undersøger funktionen hvor mange 1000 tallet består af, derefter hvor mange 900 og så videre med alle romertallene ned til 1. Derefter kalder den funktionen adding. Dette gøres ved først at dividere (/) med 1000. Resultatet af dette giver dig hvor mange 1000 tallet består af som heltal, hvilket betyder at hvis du dividere 4001, får du stadig kun 4 (4001/1000 =4 med 1 i overskud). Derefter dividerer (%) vi vores tal hvilket trækker de 4000 fra og lader os regne videre med resten.

Funktionen adding tager en streng og en integer som et argument. Herefter bliver strengen, ved hjælp af en for loop, tilføjet til en ny streng et antal gange som bliver bestemt af integeren. Herefter bliver den ny streng returnet.

Når man skal konvertere tal til romertal er det især vigtigt at huske alle reglerne for hvordan man skriver romertal. Romertal skrives lidt ligesom en addition, altså M for tusind, D for 500 og C for 100. Så hvis man skal skrive 1600 skrives det MDC. En undtagelse til denne regel er hvis man for eksempel skal skrive 900 skrives det CM altså 1000-100 i stedet for DCCCC.

Kode fordybelse for opg1:

```
	private static String adding(int times, String letter) {

		String Z = "";
		for (int i = 1; i <= times; i++) {
			Z = Z + letter;
		}
		return Z;
	}
```

Her er funktionen adding som skal kaldes med en integer og en streng som argument. Funktionen opretter en tom streng Z som den tilføjer funktionens modtagne streng til. På grund af for loopet tilføjer den, den modtagne streng så mange, til strengen Z så mange gange som integeren som funktionen blev kaldt med siger.

Dvs. at hvis funktionen bliver kaldt med M som argument i strengen og 3 som integeren, vil den returnere strengen MMM.

Test:

Input: 0

Dit input giver Nulla når det er skrevet med romertal

Hvilket er hvad vi vil forvente da 0 skrives som Nulla

Input: 3999

Dit input giver MMMCMXCIX når det er skrevet med romertal

Hvilket er hvad vi vil forvente

Input: -1

Tallet skal være større end 1, programmet afsluttes

Hvilket også er hvad vi vil forvente, eftersom romertal ikke er defineret for negative tal

Input: -0.5

```
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at Opg1.RomanNumerals(Opg1.java:12)
        at Opg1.main(Opg1.java:6)
```

Dette er også hvad vi vil forvente da programmet kun er lavet til at behandle hele tal. Dette er den error message vi får når vi gemmer en double som en int.

Input: 4000 (glæder alt over 3999)

Tallet skal være mindre end 3999, programmet afsluttes

Hvilket er hvad vi vil forvente eftersom romertal ikke går over 3999.

Konklusion

Vi har vist at vi kan ved hjælp af heltalsdivision og for løkker lave et program som "oversætter" tal fra almindelige tal til romertal. Hvis vi skal lave et program hvor vi også skal oversætte kommatal bliver vi nød til at udvide vores program væsentligt. Til gengæld kan vi med meget få ændringer få vores program til at oversætte til romertal som er større end 3999, den eneste grund til at den ikke kan gøre det er at romertal ikke er defineret for så store tal, men metoden ville være den samme.

Opgave 2 - Palindrom

Et palindrom er en tekst som set spejlvendt giver det samme, det vil sige at heh er et palindrom. Det er så sådan at man følger forskellige regler for at afgøre dette hvor den ene er at man ser bort fra specialtegn og mellemrum hvilke betyder at "he h" også er et palindrom. Det er hele teksten og ikke kun heh delen som er et palindrom.

Programmet består af 1 metode, `tjekForPalindrome.`

Metoden starter en skanner og beder brugeren om at indtaste en linje som den skal undersøge.

Den fjerner så tegn som man ikke tester når man tester for om en tekst er et palindrom, det vil sige specialtegn og tal. Der testes så for om strengen er tom, for hvis den er ville det ikke give mening at fortsætte, grunden til at testen ligger efter at vi har fjernet specialtegn er fordi disse også tæller med i længden af en streng hvis man nu havde brugt s.length().

Nu opretter vi så en ny streng som vil bestå af bruges tekst minus specialtegn, vendt om. For at opnå dette bruges hvad der hedder en StringBuilder som er en class med metoder som gør ting som reverse() og toString(). Den første del er at give den vores streng derefter vendes denne og converters som en streng som så gemmes i omvendt variablen.

Vi kan nu teste om de to strenge er ens, det vil de være hvis alle karakterer er ens og i samme rækkefølge. For at teste for dette bruger vi en metode under klassen streng som hedder equalsIgnoreCase denne vil teste om to strenge er ens uden at tage hensyn til store og små bogstaver.

Vi gemmer dette i en boolean som hedder jaEllerNej og kører så en if/else hvor vi hvis det er et palindrom og jaEllerNej derved er true printer

System.out.print("\"" + input + "\"" + " is a palindrome!");

mens den ellers printer:

System.out.print("\"" + input + "\"" + " is not a palindrome!");

Kode fordybelse for opg2:

```
if (inputNoSpecial.length() <= 0) {
       throw new IllegalArgumentException("input er tom, dette kan være fordi den med
specialtegn fjernet er blevet tom");
             }
```

inputNoSpecial indeholder brugerens streng efter den er blevet gået igennem og alle tegn som ikke er bogstaver er fjernet. Hvad vi gør her er at vi tjekker længden på strengen. Hvis længden mindre eller lig med 0 betyder det enten at brugernes input er tomt eller at brugerens input kun bestod af specialtegn. Dette program er ikke beregnet til at undersøge en streng som er tom og derfor laver vi et throw, hvor vi fortæller brugeren at der er gået noget galt og kommer med en mulig forklaring på hvad.

Test

Input: (&%"(/&¤()"%

Exception in thread "main" java.lang.IllegalArgumentException: input er tom, dette kan være fordi den med special tegn fjernet er blevet tom

at Opg2.Palindrome(Opg2.java:44)

at Opg2.main(Opg2.java:6)

Den laver en throw, hvor den fortæller brugeren at hans input er tomt. Dette er fordi et palindrom kun tester bogstaver, og ikke specialtegn som et det eneste dette input består af.

Input: 123Heh123

Enter line to check, when you press enter the program will continue: 123Heh123

123Heh123 is a palindrome!

Den kigger kun på bogstaverne og ikke på om de er store eller små

Derfor ses at heh er det samme som heh

Input: !%¤#/%aja!##"¤%)#"("#¤"

Enter line to check, when you press enter the program will continue: !%¤#/%aja!##"¤%)#"("#¤"

!%¤#/%aja!##"¤%)#"("#¤" is a palindrome!

Den kigger heller ikke på specialtegn så det bliver bare aja.

Konklusion

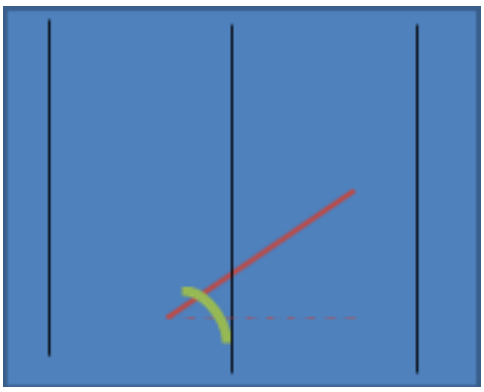
Det var her vigtigt at man tænke på hvad et palindrom er og hvordan man skulle teste for det. det var derfor nødvendigt at lave en test og derefter tage den oprindelig tekst og sende tilbage og fortælle om var et palindrom eller ej. det var jo ikke kun dele af teksten brugen skrev som evt var et palindrom men enten det hele eller intet.

Opgave 3 - Buffon's nål

Grundlaget for opgaven går ud på matematikken bag Pi og relationen til cirkler. Dette er derfor man kan beregne Pi ud fra en pind kastet random mellem streger hvis afstand imellem er double af pindens længde.

Ved enhver placering af en pind på dette underlag af streger kan altid opnås en viden om vi har ramt så pinden ligger oven på en streg eller imellem to.

Vi gør dette ved at genere to tilfældige værdier, til dette bruges biblioteket Math som har funktionen random. Denne giver en double med en værdi mellem 0 og 1 og derfor ganges så med i placeringens tilfælde afstanden mellem de to streger, og i vinklens tilfælde 180 da alle andre vinkler altså 180 og op er spejling og derfor ligegyldige.



Med disse to værdier tjekker vi så for nogen speciel tilfælde såsom at pinds ender stå på stregen hvorfor vi så bare kalder det et hit uden at lave yderligere beregninger, desuden hvis vinklen er 180 eller 0 betyder det også at vi nemmere kan beregne den længde vi skal finde for at bestemme om vi rammer en streg eller ej.

Kode fordybelse for opg3:

```
for (int i = 1; i <= antalKast; i++) {
                    double a = Math.random() * lengthBetweenLines; // placeringen
                    double b = Math.random() * 180; // vinklen.
                    double restLength = 0;

                    if (a == lengthBetweenLines || a == 0) {
                           hits++;
                    } else {
                           if (b == 180 || b == 0) {
                                  restLength = lengthOfStick;
                           } else {
                                  b = 180 - 90 - b;
                                  restLength = (Math.sin(Math.toRadians(b))
                                  * lengthOfStick)
                                  / (Math.sin(Math.toRadians(90)));
                           }
                           if ((a + restLength) >= lengthBetweenLines
                                        || (a + restLength) <= 0) {
                                  hits++;
                           }
                    }
             }
```

Det ses at der er tre veje videre.
Hvis den første er sand kan det øge hits med 1 og gå til næste iterationer.
Hvis den første er falsk og den næste er sand kan det hoppe til den sidste uden at bruge pakkerne Math.sin() og Math.toRadians().
Disse tre veje tager ikke den samme tid at løbe i gemmen hvorfor vi netop har dem, da vi gerne vil spare tid og kræfter.

Efter dette for loop har kørt de antal iterationer brugeren ønskede går den videre og finder en tilnærmelse af Pi ved at dividere iterationer med antal hits, da variablen ”hits” bliver castet til en double vil vi få et decimal tal.

Test

Test kørt hvor antal iterationer er sat til 0 eller mindre:

```
Hvor mange kast vil du foretage? 0
Exception in thread "main" java.lang.IllegalArgumentException: Ingen kast bedt om
	at Opgave3.antalKastMedBuffonsNeedle(Opgave3.java:22)
	at Opgave3.main(Opgave3.java:8)
```

Det ses at den vil give en fejlmeddelelse, som jeg selv har skrevet som siger at ingen kast bedt om. Dette er selvfølgelig fordi det ikke give mening at køre resten 0 gange.

Test kørt hvor antal iterationer er sat til 1/2:

```
Hvor mange kast vil du foretage? 0.5
Exception in thread "main" java.util.InputMismatchException
	at java.util.Scanner.throwFor(Unknown Source)
	at java.util.Scanner.next(Unknown Source)
	at java.util.Scanner.nextInt(Unknown Source)
	at java.util.Scanner.nextInt(Unknown Source)
	at Opgave3.antalKastMedBuffonsNeedle(Opgave3.java:18)
	at Opgave3.main(Opgave3.java:8)
```

Den kan ikke hente værdien som den skal bruge videre i programmet da denne ikke findes når den kun er sat til at lede efter integers.

Test kørt hvor antal iterationer er sat til 1:

```
Hvor mange kast vil du foretage? 1
Exception in thread "main" java.lang.IllegalArgumentException: Ingen hits
	at Opgave3.buffonsNeedle(Opgave3.java:64)
	at Opgave3.antalKastMedBuffonsNeedle(Opgave3.java:25)
	at Opgave3.main(Opgave3.java:8)
```

Der er en stor chance for at man vil få denne meddelelse hvis man køre 1 eller i hvert fald få gange. Den kommer når man under sit for loop aldrig øgede hits og derfor aldrig ramte en streg. Havde man fortsat koden ville man være kommet til at dividerer med 0 hvilke ikke er muligt.

Test kørt hvor antal iterationer er sat til over 2^31-1:

```
Hvor mange kast vil du foretage? 99999999999999999999999999999999999999
Exception in thread "main" java.util.InputMismatchException: For input string:
"99999999999999999999999999999999999999"
	at java.util.Scanner.nextInt(Unknown Source)
	at java.util.Scanner.nextInt(Unknown Source)
	at Opgave3.antalKastMedBuffonsNeedle(Opgave3.java:18)
	at Opgave3.main(Opgave3.java:8)
```

Det som sker her er at den leder efter en int men den finder et tal som er større og kalder derfor en fejl som siger InputMismatchException. Den kommer ikke videre i koden da den ikke kan finde den værdi den skal bruge for at gøre dette.

Konklusion

Det sås at vi var i stand til at finde Pi med ret høj nøjagtighed og kunne sagtens med nogle ændringer få flere decimaler, dog ville den tid man skulle bruge på beregning også stige kraftigt hvorfor man skal have så mange steder i programmet hvor en lille test kan spare programmet for at løbe en masse andet i gennem.