# Hjemmeopgave01

## Kursus 02101
## Indledende Programmering

Lukas Villumsen (s144451)
Sebastian Nyholm (s144434)

7. oktober 2014

## Indhold

# 1 Opgave 1.1 - Password

Password class'en er lavet med en enkelt public metode, så den kan blive brugt udefra. Metoden er lavet, så den returner en boolean, altså false eller true, da metoden kun skal afgøre om det indtastede password overholder de givne regler.
Vi har lavet metoden, så den gennemgår alle reglerne en efter en og hvis den støder ind i en fejl undervejs, returner den false med det samme, så man undgår at gennemgå resten af koden, da passwordet allerede har fejlet.
Alle reglerne bliver tjekket af if statements, men de fleste steder skal alle tegn i passwordet tjekkes, hvilket bliver gjort ved hjælp af en for lykke.

```
for (int i = 0; i <passLength; i++)
```

I for lykken er der if statements inden i, som afgør om passwordet holder og hvis det fejler eller holder, alt efter hvad reglen er, breaker lykken, så man ikke behøver køre den hele vejen igennem.[1]

[1]Se bilag - Kildekode: Password.java

# 2 Opgave 1.2 - NextPrime

Idéen bag programmet er en simpel generering af primtalsrækken med den enkelte forskel, at der ikke køres igennem alle positive integers. Den givne *integer* (heltal) $n$ udgør således begyndelsesværdien for primtalsgenereringen, hvorefter det først fundne primtal printes til konsollen som svar. Primtalsalgoritmen brugt, I dette tilfælde, går ud på først at:

- Ignorere alle multiplum af 2 - Der findes ingen lige primtal støre end 2.
- Hvis $n < 2$, så er det første primtal efter $n$ lig 2.
  - Primtal er kun defineret for **positive** heltal.
- Kun heltallet $n = 2$ vil kunne resultere i udskrivningen af primtallet 3.

Den mere bærende del af algoritmen, og dermed programmet, består i at teste for divisorere $d < \sqrt{n}$. Grundet den måde faktorere af $n$ gentages efter $\sqrt{n} * \sqrt{n}$, er der ingen grund til at iterere yderligere end til dette punkt $(d < \sqrt{n})$. Dette kommer af at multiplikation er *kommutativt*:

$$36 = 18 * 2 = 12 * 39 * 4 = \sqrt{36} * \sqrt{36} = 4 * 9 = 3 * 12 = 2 * 18 = 36$$

Vi tester altså $n$ for alle *ulige* divisorere $d < \sqrt{n}$, hvis ingen findes er $n$ et primtal.[2]

[2]Se bilag - Kildekode: NextPrime.java

# 3 Opgave 1.3 - AccessControl

Dette program skulle laves med egen main metode, da den kan gribes an på mange måder. Vi valgte at lave en metode for hver enkelt del af systemet. Vi endte derved op med følgende metoder:

- main()
- signIn()
- logIn()
- online()
- changePassword()
- logOff()
- shutDown()

Det første der sker når man åbner programmet er at man fra mainmetoden bliver sendt over i signIn() metoden, hvor man bliver bedt om at lave en bruger. Når man har lavet en korrekt bruger bliver man sendt videre til logIn(). Her kan man logge ind, hvis man skriver de rigtige oplysninger og ellers bliver man bedt om at indtaste dem igen. Hvis det er korrekt, bliver man sendt videre til online() metoden, hvor man kan vælge de tre muligheder. Alt efter hvad man vælger, bliver man sendt videre til den tilhørende metode. Hvis man vælger at ændre password, bliver man sendt videre til changePassword(), hvor man har mulighed for at ændre passwordet. Hvis man indtaster noget korrekt, bliver man sendt tilbage til online() metoden. Hvis man vælger log ud, bliver man sendt til logOff(), som sender en videre til logIn(). Her kunne man godt have udeladt logOff() metoden, men i tilfælde af en videre udviklen, valgte vi at lave den. Hvis man vælger at lukke systemet, bliver man sendt til shutDown(), som lukker ned for systemet. Her kunne man også gøre som før og slukke systemet direkte i online() metoden, men af samme grund som før, valgte vi at lave metoden for sig selv.[3]

[3]Se bilag - Kildekode: AccessControl.java

# 4 Opgave 1.4 - Particles

## a)

I denne opgave benyttes objektvariablen *Point* med henblik på at overskueliggøre koordinatsættet $(x_n, y_n)$ for partiklen $P_n$.
Der gøres brug af den importerede metode `Random()`, hentet fra java's utility pakke, til hhv. initialiseringen af de tre partikler, $P_1, P_2, P_3$, samt ved den tilfældige forøgelse (eng. *increment*) i hvert af partiklernes koordinater. Selve den 'aktive' del af programkoden eksekveres inden for et `for loop`, hvor metoderne:

- `Point.getX()`
- `Point.getY()`
- `Point.setLocation()`

anvendes til bestemmmelsen af partiklernes nye position på det ikke-synlige $n \times n$ grid. Loopet itereres $t$ gange hvorved $P_n$`.getX()` såvel som $P_n$`.getY()` bliver inkrementeret med hvert sit særskilte tilfældige heltal i intervallet $[-s, s]$ for hver iteration.[4]

## b)

Følger foregående opgave, dog med få justeringer:

- Fjernet én af partiklerne, $P_3$.
- Tilføjet et *nested* `for loop`

Når de to resterende partikler ved tilfældig inkrementering opnår ens koordinatsæt vil det nye loop træde i kraft. Det nye loop kører videre på det oprindelige `for loop`, med samme iterationsbetingelser, og derved terminerer begge loop efter endt udførelse på det *nested* `for loop`.[5]

[4] Se bilag - Kildekode: Particles.java
[5] Se bilag - Kildekode: ParticlesWithCollide.java