Kravspesifikasjon

Prosjektoppgave i faget Programutvikling (DATS1600 / ITPE1600), 27. mars - 20. mai 2015.

Gruppeinformasjon

Vi har valgt å ta for oss oppgave 1: Forsikring.

Navn	Studentnummer	Studielinje	Klasse
Martin Mellum Jackson	s236629	Informasjonsteknologi	INFORMATIK14HA
Baljit Singh Sarai	s169947	Informasjonsteknologi	INFORMATIK14HA
Adrian Siim Melsom	s236308	Dataingeniør	HINGDATA14HA

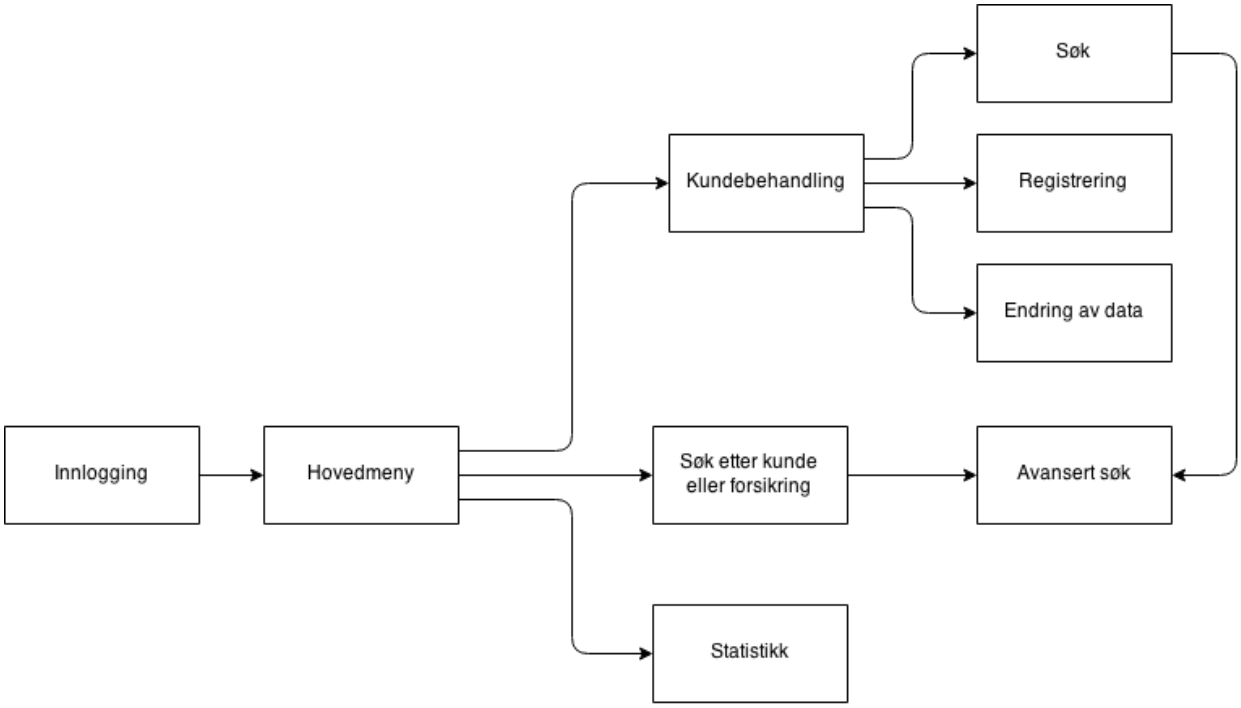
Bruk av programmet

Brukere

Som vi har tolket oppgaven vil brukerne av programmet være ulike personer som jobber i et forsikringsselskap. Vi ser for oss at ikke alle brukerne jobber i samme type stilling, og derfor behøver tilgang til ulike deler av programmet. For eksempel er et av kravene til oppgaven å kunne tilby statistikk om de forskjellige dataene som selskapet lagrer. Denne informasjonen vil være mest relevant for ledelsen i selskapet. Andre oppgaver, som registrering av nye kunder og forsikringer, vil derimot være mest aktuelle for rådgivere og konsulenter.

Use-case

Følgende diagram viser hvordan programmet vil kunne brukes.



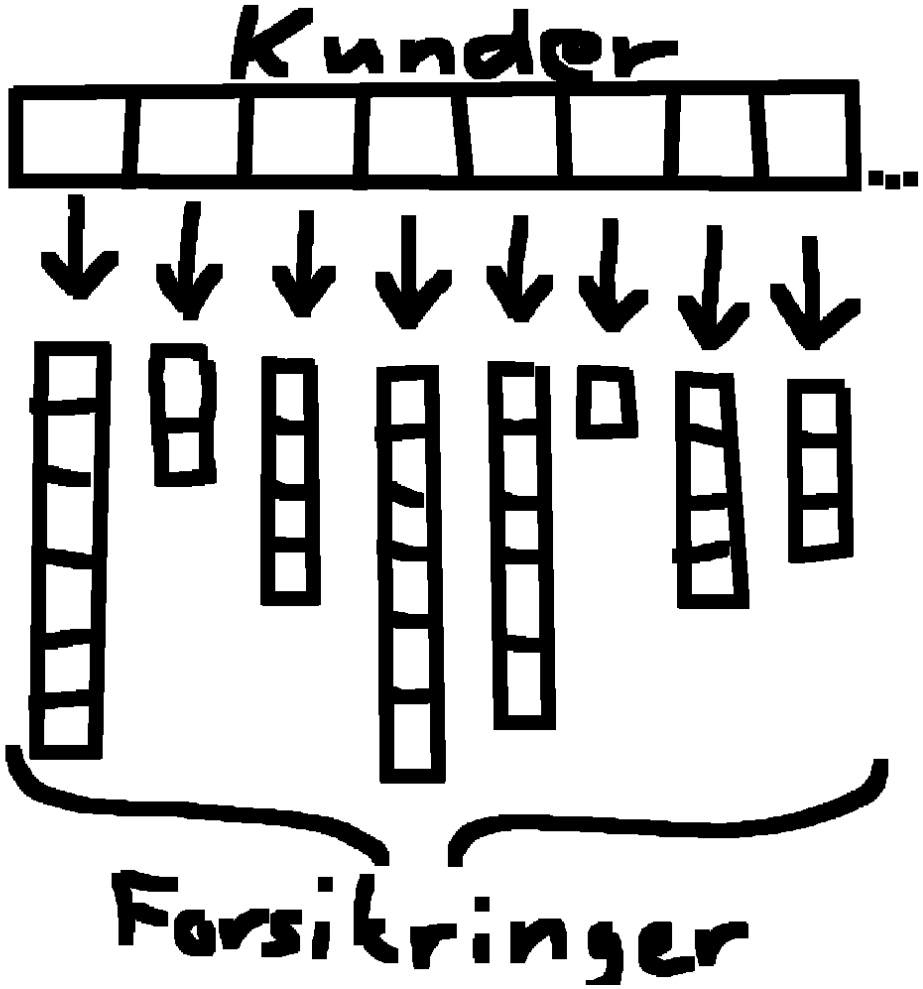
Kundebehandlingsdelen og enkelt søk er operasjoner som blir prioritert, fordi disse gir grunnlag for både statistikk og avansert søk.

På dette stadiet i oppgaven planlegger vi at det skal være mulig å hente ut statistikk over kunder og forsikringer, men dersom tiden tillater det vurderer vi også å legge inn statistikk over de ansatte i selskapet. Dette fører til et krav om forskjellige brukerrettigheter i programmet, ettersom en vanlig ansatt ikke uten videre skal kunne se hva kollegene foretar seg.

Datastruktur

Til datastrukturen vår kommer vi til å bruke Collections-rammeverket. Det er mange forskjellige implementasjoner å velge blant, og vi har vurdert hvilke av dem som passer best for våre formål.

Vi har valgt å benytte oss av en ArrayList til lagring av kundene, og til å lagre hver kundes forsikringer. De ansatte i bedriften vil lagres i en separat ArrayList. Denne strukturen illustreres i skissen nedenfor. Pilen indikerer at elementet inneholder en peker til listen det peker på.



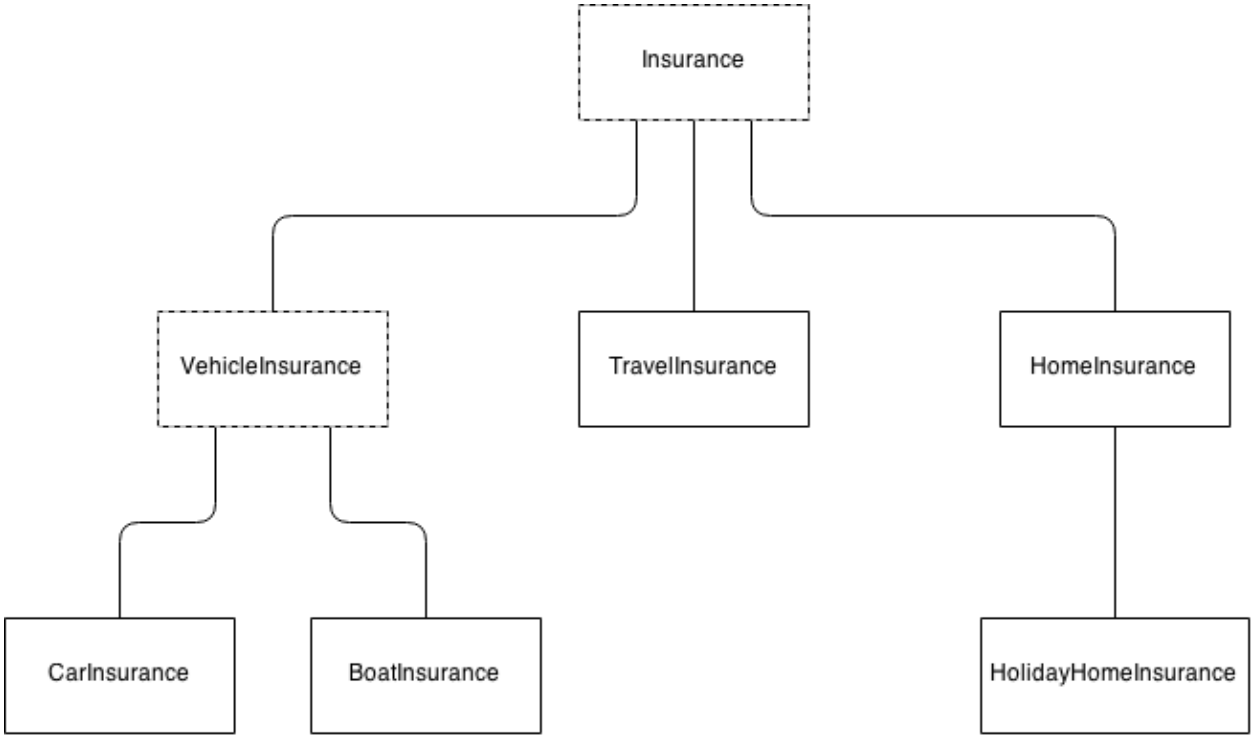
Hvordan vi har argumentert for denne beslutningen fremgår i dokumentet Prosessdokumentasjon.

Klasser

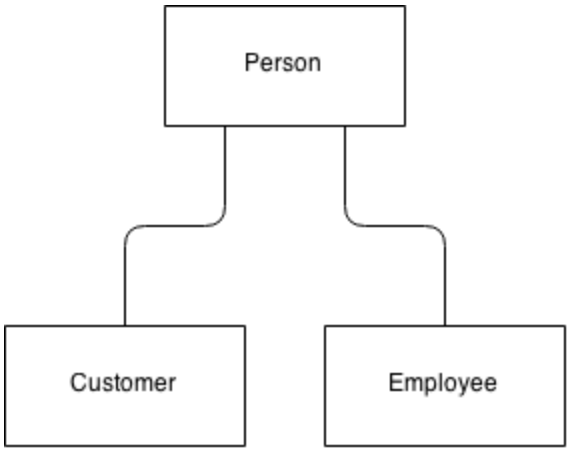
Klassehierarkier

Følgende diagrammer illustrerer klassehierarkiene i koden vår. De øvre klassene med linjer ned til andre klasser er superklasser. Stiplete linjer indikerer en abstrakt klasse.

Forsikringsklassene



Personklassene



Grunnen til at Personklassen ikke er abstrakt, er at vi ønsker å bruke den som datafelt i klassen ClaimAdvice, og at det ikke skal lagres noe ekstra om denne persontypen. Dersom vi senere finner ut at dette ikke er tilfelle, vil det være enkelt å implementere en subklasse kalt Witness.

Klasseinnhold

Nedenfor følger en oversikt over hva som skal lagres i programmets kjerneklasser. Hvordan vi velger å implementere dette i form av datafelter i klassene vil ikke nødvendigvis komme tydelig frem. Noen av tingene vi vil lagre kommer til å bli egne klasser, mens andre kommer til å være av enkle typer som tekst og heltall. Videre kan det hende at vi senere i utviklingen innfører nye klasser som kan komme til å endre på strukturen vi presenterer her.

NB: For klasser i et klassehierarki vil all informasjon som allerede er lagret i en eventuell superklasse ikke tas med.

Person

- navn
- personnummer
- telefonnummer
- adresse

Customer

Arver fra Person.

- dato for opprettet kundeforhold
- fakturaadresse
- alle forsikringer
- årlig premie for alle forsikringer
- totalkunde?
- utbetalte erstatninger

Employee

Arver fra Person.

- stilling
- rettigheter (dersom vi innfører statistikk over ansatte i programmet)

ClaimAdvice

- dato for skaden
- skadenummer
- type skade
- for biler: skademeldingsskjema
- beskrivelse av skaden
- bilder av skaden (hvis det foreligger)
- kontaktinformasjon til eventuelle vitner til ulykken
- takseringsbeløp av skaden

- utbetalt erstatningsbeløp (kan være lavere enn det takserte skadebeløpet)

Settlement

- dato for betaling
- beløp
- erstatningsårsak

Insurance

- forsikringsnummer
- ansatt som registrerte forsikringen
- årlig forsikringspremie
- dato for opprettet forsikring
- dato for avsluttet forsikring
- forsikringsbeløp
- forsikringsbetingelser: informasjon om hva forsikringen dekker
- skademeldinger

VehicleInsurance

Arver fra Insurance.

- eier
- registreringsnummer
- type
- modell

CarInsurance

Arver fra VehicleInsurance

- registreringsår (første gang)
- kjørelengde pr. år i antall km
- pris pr. kilometer
- bonus

BoatInsurance

Arver fra VehicleInsurance.

- lengde i fot
- årsmodell
- motortype
- motorstyrke i HK

HomeInsurance

Arver fra Insurance.

- adresse (kan være forskjellig fra kundens fakturaadresse)
- byggeår
- boligtype
- byggemateriale
- standard

- antall kvadratmeter
- forsikringsbeløp for bygning
- forsikringsbeløp for innbo

HolidayHomeInsurance

Arver fra HomeInsurance.

- utleie

TravelInsurance

Arver fra Insurance.

- informasjon om hvem forsikringen gjelder
- område (hvor forsikringen gjelder)

I tillegg skal vi blant annet skrive registerklasser som behandler disse, og klasser som samhandler med disse registrene og brukergrensesnittet. Vi tar ikke med disse i oversikten ovenfor fordi ingen av deres tilstander lagres når programmet avsluttes.

Moduler

Brukergrensesnitt

Denne modulen samler alt som har med brukergrensesnitt å gjøre på ett sted. Dette var et naturlig valg, ettersom brukergrensesnittet ikke har noe å gjøre med det som skjer i "backend"; det skal bare presentere dataene.

Kontroller

Kontroller-modulen håndterer alt av input-validering og liknende.

Data

Data-modulen vil inneholde alle kjerneklassene, samt registerklasser som samler data fra disse.

Statistikk

Å ha en egen modul for statistikk var enda et naturlig valg. Klassene i denne modulen skal motta samlinger med data fra Data-modulen og returnere resultater som kan presenteres i Brukergrensesnitt-modulen.

Ved å separere hentingen av data fra kjerneklassene, og koden som tar i bruk disse til å lage statistikk, skaper vi et skille i koden som vil gjøre det enklere å fokusere på mindre og mer håndterlige biter av programmet.

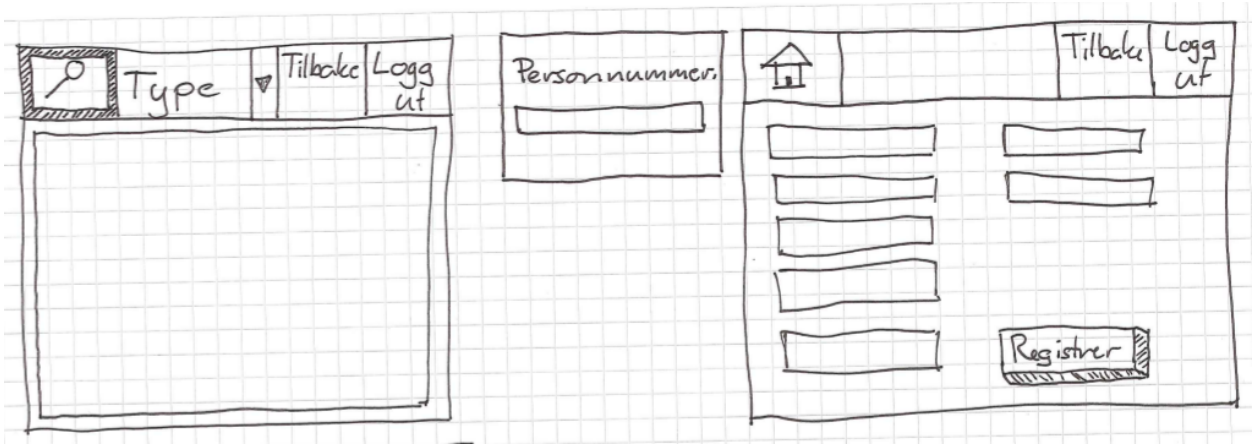
Brukergrensesnitt

Hovedmeny

På hovedmenyen er det planlagt å være tre knapper, og når man klikker på disse leder de til nye sider: “Kundebehandling”, “Finn en kunde eller forsikring” og “Statistikk”.

Kundebehandling

Skissen nedenfor illustrerer kundebehandlingssiden, og hvordan registrering av en ny forsikring eller kunde vil foregå. Vinduet helt til venstre skal være hovedvinduet for kundebehandlingsdelen. Derfra skal man kunne søke, velge type forsikring som skal registreres, og gå tilbake eller ut av programmet.



Når man har valgt type forsikring, blir man sendt til et nytt vindu der man blir bedt om å skrive inn personnummeret til kunden man skal registrere forsikringen på. Om kunden allerede eksisterer blir man sendt videre til et skjema hvor man fyller ut informasjon om forsikringstypen som ble valgt på starten. Hvis kunden ikke eksisterer, blir man først sendt til et skjema hvor man registrerer kunden. Vi har valgt at man ikke skal kunne registrere en kunde uten forsikring, og det er derfor ikke mulighet for det her. Om man avbryter forsikringsregistreringen etter å ha fylt ut kundeskjemaet, vil kunden ikke bli registrert.

Finn en kunde eller forsikring

Denne siden skal inneholde to søkemenyer: Én for kunder og én for forsikringer. Dersom tiden tillater det, skal det også lages en link til siden for avansert søk.

Statistikk

Statistikksiden skal inneholde forskjellige søkefelt og rullegardinmenyer som gir størst mulig fleksibilitet i hentingen av data. Dersom vi har kapasitet til det, ønsker vi å legge til en meny for statistikk over ansatte. Denne skal, som nevnt tidligere, bare utvalgte ansatte ha tilgang til.

Kodestandard

Språk

All koden kommer til å bli skrevet på engelsk. Vi har valgt dette fordi de norske bokstavene æ, ø og å tidligere har skapt problemer for oss når koden deles med andre. I tillegg er engelsk mer gunstig med tanke på navngiving av metoder på CamelCase-form, ettersom det er et språk med betydelig større bruk av orddeling enn norsk.

Kommentarer

Vi kommer til å skrive dokumentasjonskommentarer til alle klasser og deres metoder. Slike kommentarer vil inneholde vesentlig informasjon, i tillegg til en forklaring på hva klassen tilbyr eller hva en metode gjør, hva den tar som argument og hva den returnerer. Når vi kommer til dokumentasjonsfasen vil det være mulig å bruke kommentarene vi har skrevet til å generere en Javadoc-fil som samler og organiserer dem.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
       try {
           return getImage(new URL(url, name));
       } catch (MalformedURLException e) {
           return null;
       }
}
```

Eksempel på dokumentasjonskommentar hentet fra
http://www.oracle.com/technetwork/articles/java/index-137868.html.

Kodestil

Mange av punktene under er hentet fra Google sine retningslinjer om kodestil i Java: https://google-styleguide.googlecode.com/svn/trunk/javaguide.html.

Formattering

- Ingen linjeskift før åpnende krøllparenteser.
- Linjeskift etter åpnende krøllparenteser, og lukkende krøllparenteser hvis disse avslutter en setning eller en metodekropp. Det er for eksempel ikke linjeskift etter krøllparentesen dersom den er etterfulgt av en **else**.
- Bruke krøllparenteser selv der det er valgfritt, som for eksempel når en **if**-blokk bare inneholder én linje.
- Mellomrom mellom nøkkelord som **if**, **for** og **while** og deres påfølgende parenteser. Dette i motsetning til metodekall, som ikke skal ha mellomrom mellom metodenavnet og parentesen. Vi har valgt dette for å tydelig skille mellom Javas nøkkelord og metoder, i tillegg til at det ser ryddigere ut.
- Til innrykk brukes 4 mellomrom i stedet for tabulator.
- Holde antall tegn per linje til maksimalt 80.
- Én tom linje mellom hver metode i en klasse, og inne i metoder der det skaper logiske skiller i koden.

Navngiving

- Klassenavn skrives i *UpperCamelCase*.
- Metode- og variabelnavn skrives i *lowerCamelCase*.
- Konstanter skrives i *CONSTANT_CASE*.

Struktur

- **package**- og **import**-setninger på toppen av filen.
- Dersom en klasse har flere konstruktører eller metoder som er overloadet, skal disse forekomme rett etter hverandre i koden.

Annet

- Ved kall på statiske metoder eller aksessering av statiske felter skal klassenavn brukes, ikke en instans av klassen.

Filklasser

Som nevnt i avsnittet om datastrukturer, vil alle dataene lagres i samlingen med kunder og samlingen med ansatte. Det er dermed disse som vil være ønskelige å lagre. Vi har valgt at alle klassene som inngår i datastrukturen skal serialiseres, slik at alt vi behøver å skrive til fil er de to Collection-objektene. Videre skal disse befinne seg i to ulike filer, slik at om programmet bare skulle ha behov for de ansatte, skal bare filen med ansattlisten lastes inn. Filene skal være av filtypen .ser, som er standard for filer som inneholder serialiserte objekter i Java.

Vi ønsker også at programmet skal skrive vesentlige hendelser til en loggfil, som skal være av typen .txt. Eksempler på hendelser vi vil lagre er opprettelse av nye kunder og forsikringer, i tillegg til oppdatering av disse. Hver melding i loggfilen skal inneholde tidspunktet for hendelsen og en kort beskrivende beskjed. Grunnen til at vi valgte filtypen .txt er at denne skal kunne leses uten å måtte åpne programmet vårt.

Arbeidsfordeling

Vi har bestemt oss for en noe løs arbeidsfordeling. Fordelen med dette er at alle blir involvert i hverandres andres arbeid, og raskt kan hjelpe til eller ta over ved behov. Imidlertid har vi fordelt ansvarsområder for å sikre at alle bidrar likt, og at tidsplanen følges.

Martin

Martin har hovedansvar for dokumentasjon og rapporter, i tillegg til kjerneklassene i programmet. Han vil assistere Baljit i arbeidet med klassene som bruker disse.

Adrian

Brukergrensesnittet blir utviklet av Adrian, med innspill fra resten av gruppen.

Baljit

Baljit har ansvar for klassene som handler om bruk av kjerneklassene. Disse skal kunne hente ut diverse data og organisere dem på forskjellige måter. Det er ikke mange slike klasser, men til gjengjeld forventes de å være mye større og kompliserte. Dette fører til at arbeidsmengden blir tilnærmet lik de andres.

Fremdriftsplan

Planleggingsfase, 27. mars - 13. april

- Skrive kravspesifikasjon som leveres innen 13. april kl. 15:00.

Implementasjonsfase, 14. april - 10. mai

14. - 21. april

- Være så godt som ferdig med brukergrensesnitt
- Bli ferdig med kjerneklasser
- Være godt i gang med “manager”-klassene

22. april - 1. mai

- Bli ferdig med “manager”-klassene og lagring
- Være godt i gang med statistikk og søk

2. - 10. mai

- Bli ferdig med statistikk og søk
- Finpusse programmet
- Eventuelt legge til ekstra funksjonalitet

Dokumentasjonsfase

- Mer finpuss der det trengs
- Skrive dokumentasjonskommentarer der det mangler og generere Javadoc fra disse

Levering, 20. mai

- Levere programmet og dokumentasjon innen kl. 15:00.

Teknisk informasjon

- **Javaversjon:** 1.8
- **Utviklingsverktøy:** Eclipse og IntelliJ