
EASYDEV

GRUPPE TUXMIN

Espen Zaal s198599
Lukas Larsed s198569
Petter Knagenhjelm Lysne s198579

13. november 2014



Innhold

1	Introduksjon	5
1.1	Om rapporten	5
1.2	Formål	5
1.3	Tolkning av oppgaven	6
1.3.1	Brukergrensesnitt	6
1.3.2	Boligøskere	6
1.3.3	Meglere	7
1.3.4	Historikk	7
1.3.5	Utleiere	7
1.3.6	Boliger	7
1.4	Mål	8
1.5	Tekniske detaljer	8
1.5.1	Utviklingsmiljø	8
1.5.2	Krav til programvare	8
1.5.3	Versjonshåndtering	9
2	Prosessdokumentasjon	10
2.1	UML	10
2.2	Arkitektur	11
2.3	Arv og polymorfisme	11
2.3.1	Swing komponenter	11
2.3.2	Pakkefordeling	12
2.3.3	Konstanter og enum	12
2.4	Arbeidsfordeling	13
2.5	Utfordringer	13
2.5.1	Kontrollerhierarki	13
2.5.2	Serialisering	14
2.5.3	Tabellmodell	14
2.5.4	Layout-managere	14
2.5.5	Konflikter i GIT	14
2.5.6	Objektreferanser	15
2.5.7	Swing og portabilitet	15
2.5.8	Generiske metoder og klasser	16
3	Produktdokumentasjon	17
3.1	Pakker	17

3.2	Start og avslutting	18
3.3	Serialisering	19
3.4	MVC	20
3.4.1	Generelt om den strukturelle oppbyggingen	20
3.4.2	Oppstart av kontrollere og brukergrensesnitt	21
3.4.3	Oversikt over hvilken kontrollere som h�rer til hvilken GUI-klasse	22
3.5	Datastruktur (Model)	22
3.5.1	Valg av datastruktur	22
3.5.2	Datatyper	22
3.6	Brukergrensesnitt (View)	24
3.6.1	Oppstart av brukergrensesnittet	24
3.6.2	Bruk av arv i brukergrensesnittet	25
3.6.3	Oppbyggingen av arkfanene	25
3.6.4	Registreringsvinduer	26
3.7	Kontroller (Controller)	27
3.7.1	Generelt om kontrollermilj�et	27
3.7.2	Hovedkontrollerne	28
3.7.3	MainController.java	28
3.7.4	ControllerToppPanelMegler.java	28
3.7.5	ControllerToppPanelAnnonse.java	29
3.7.6	ControllerBunnPanel.java	29
3.7.7	ControllerOutput.java	29
3.7.8	ControllerTabell.java	30
3.7.9	Kontrollerne for registreringsvinduer	38
3.7.10	Innloggingskontroller	41
3.7.11	Tastatursnarveier: ControllerKeyBindings	42
3.7.12	Lyttere mellom forskjellige kontrollere i programmet	42
3.8	S�k	43
3.8.1	Meglers�k	44
3.8.2	Annonsefilter	45
3.9	Bilder	47
3.9.1	Bildeklasser	47
3.9.2	Lagring av bilder	48
3.9.3	Visning av bilder	49
3.9.4	Sletting av bilder	49
3.10	Konstanter og Enum	52
3.10.1	RegexTester.java	52
3.10.2	Konstanter.java	53
3.10.3	GUI konstanter	53
3.10.4	Enum	54
3.11	Swing komponenter	55
3.11.1	AbstractPanel.java	55
3.11.2	MainPanel.java	56
3.11.3	AbstraktArkfane.java	56
3.11.4	TopPanelMegler.java	57
3.11.5	CustomSubPanel.java	58
3.11.6	CustomJTextField.java	58

3.11.7	CustomJButton.java	59
3.11.8	ComboDataVelger.java	60
3.12	Visuelle detaljer	61
3.12.1	Ikoner	61
3.12.2	Presentasjon	61
3.12.3	Tabell	61
3.12.4	Grafisk tema	63
4	Brukerveiledning	64
4.1	Forord	64
4.2	Bolisøker	65
4.2.1	Filterpanel	65
4.2.2	Resultattabell	66
4.2.3	Visningspanel	67
4.2.4	Forespørsel/søknad	67
4.3	Megler/administrasjon	69
4.3.1	Pålogging	69
4.3.2	Menyer	69
4.3.3	Søkepanel	70
4.3.4	Resultattabell	71
4.3.5	Visningspanel	72
4.3.6	Utleieradministrasjon	73
4.3.7	Annonseadministrasjon	74
4.3.8	Boligadministrasjon	75
4.3.9	Kontraktadministrasjon	76
4.3.10	Sletting	76
4.3.11	Hurtigtaster/hotkeys	76
5	Testrapport	77
5.1	Testing utført	77
5.1.1	Testing underveis	77
5.1.2	Funksjonstest av ferdig program	77
5.2	Begrensninger	79
5.2.1	Søk	79
5.2.2	Generell brukeropplevelse	79
5.3	Kjente feil (Bugs)	80
5.3.1	Feil i fritekstsøk	80
5.4	Forbedringer	80
A	Diagram	82
B	Fremtaging av GUI	88
B.1	Revisjon 1	88
B.2	Revisjon 2	92

Eksempler

2.1	Teste for objekttype	15
3.1	Oppstart av programmet.	18
3.2	Serialisering og skriving av data.	19
3.3	Innlesing av serialisert data.	19
3.4	<code>ControllerTabell.java</code>	26
3.5	<code>ControllerBunnPanel.java</code>	29
3.6	Slettefunksjonalitet i tabellen ved å trykke Delete på tastaturet.	31
3.7	Lytteren som finner valgt rad i tabellen.	31
3.8	Metoden <code>settCelleRenderer</code> fra <code>VenstrePanel.java</code>	32
3.9	Metoden <code>resizeKolonneBredde</code> fra klassen <code>VenstrePanel.java</code>	33
3.10	Metoden <code>settInnDataITabell</code> i <code>ControllerTabell.java</code>	33
3.11	Lytter for museklikk i output-vinduet.	35
3.12	Hendelse ved dobbelklikking på et objekt i tabellen.	36
3.13	Menyvalg ved høyreklikk i tabellen.	36
3.14	Funksjonaliteten til to av menyvalgene i pop-up menyen.	37
3.15	<code>AbstractControllerRegister.java</code>	38
3.16	<code>ControllerRegistrerBolig.java</code>	39
3.17	<code>ControllerRegistrerBolig.java</code>	39
3.18	<code>ControllerRegistrerBolig.java</code>	40
3.19	<code>ControllerRegistrerBolig.java</code>	40
3.20	Oppsett av tastatur snarveier.	42
3.21	<code>ControllerToppPanelAnnonse.java</code>	42
3.22	Setter lytter fra <code>MainController.java</code>	43
3.23	<code>ControllerToppPanelAnnonse.java</code>	43
3.24	Oversikt over <code>Searchable</code> interface	44
3.25	<code>toSearch()</code>	44
3.26	Iterasjon med <code>Searchable</code>	45
3.27	<code>AnnonseFilter.java</code> : Konstruktør	45
3.28	<code>AnnonseFilter.java</code> : Filtreringsrekkefølge	46
3.29	<code>AnnonseFilter.java</code> : privat filtreringsklasse	47
3.30	<code>BoligBilde.java</code> : Innlesning av bildefil	48
3.31	<code>BoligBilde.java</code> : Endring av opplastet bildestørrelse	48
3.32	<code>BoligBilde.java</code> : Lagring av et nytt eller tillegsbilde for en bolig	48
3.33	Regexstreng for gateadresse og husnummer.	52
3.34	Private regex test metode.	52
3.35	Static regex metode til tilhørende regex mønster streng.	53

3.36	Noen av static konstanter som brukes i Konstanter klassen.	53
3.37	Enum klasse for vindustørrelser	54
3.38	Utsnitt fra konstantklasse med static variabler for programikoner.	54
3.39	Konstruktør i <code>MainPanel.java</code>	56
3.40	Konstruktør til <code>AbstraktArkfane</code>	56
3.41	De forskjellige konstruktørene i <code>CustomJButton</code>	59

Figurer

3.1	MainController: Programoppstart	24
3.2	Utsnitt fra boligbehandlingsvindu # 1.	49
3.3	Utsnitt fra boligbehandlingsvindu # 2.	50
3.4	Bildevisning	50
3.5	Standardbilde	51
3.6	Komponenter i AbstraktArkafane	58
3.7	GUI komponenter i meglerpanel	59
3.8	Forsjellige tilstand av CustomJPanel	59
3.9	ComboDatoVelger.java	60
3.10	Applikasjons og vinduikoner	61
3.11	HTML presentasjon av et boligobjekt.	62
3.12	Visning og markering i tabell	62
3.13	Applikasjons og vinduikoner	63
4.1	Boligsøkende.	64
4.2	Filterpanel.	65
4.3	Resultattabell.	66
4.4	Visningspanel.	67
4.5	Forespørsel og krav fra utleier.	67
4.6	Registrering for leietaker.	68
4.7	Pålogging for megler.	69
4.8	Søkepanel.	70
4.9	Resultattabell.	71
4.10	Visningspanel.	72
4.11	Utleieradministrasjon.	73
4.12	Annonseadministrasjon.	74
4.13	Boligadministrasjon.	75
4.14	Kontraktadministrasjon.	76
A.1	UML	83
A.2	Brukercase	84
A.3	MVC - første utkast	85
A.4	Kontroller og GUI	86
A.5	Tabellmodell og output	87
B.1	Fremtaging: Velkomstbilde	88

B.2	Fremtaging: Boligsøk	89
B.3	Fremtaging: Meglersøk	89
B.4	Fremtaging: Megler GUI	90
B.5	Fremtaging: Boligregistrering	90
B.6	Fremtaging: Saksbehandling	91
B.7	Fremtaging 2: Søkervindu	92
B.8	Fremtaging 2: Meglervindu	93

Tabeller

2.1	Arbeidsfordeling mellom gruppemedlemmer	13
3.1	Oversikt over kontrollerne og deres tilhørende GUI-klasser	22
5.1	Testrapport – Linux (Ubuntu 12.04)	78
5.2	Testrapport - Windows 8.1	79

Kapittel 1

Introduksjon

1.1 Om rapporten

Denne rapporten består av flere kapitler som kan leses hver for seg og som har hvert sine formål.

Introduksjonen vil gå gjennom litt av forutsetningene for oppgaven, målene vi har satt oss, tolkningen av oppgaven og valgene vi har tatt på bakgrunn av det.

Prosessdokumentasjonen vil beskrive aspektet ved arbeidet vårt. Hvordan vi kom sammen som en gruppe, bestemte oss for fremgangsmåte og utfordringene vi har stått overfor underveis.

Produktdokumentasjonen er av det veldig tekniske slaget. Det er gitt mange illustrasjoner og kodeeksempler på utvalgte metoder og funksjonalitet, slik at det skal være overkommelig for utenforstående å sette seg inn i programmet.

Testrapporten vil beskrive de tester vi har utført, hvordan vi har utført dem og hvilke resultater de gav.

Brukerdokumentasjonen vil både være inkludert i dette dokumentet, samt som et frittstående dokument. Den dokumentasjonen vil gi brukeren oversikt over hvordan en bruker programmet og hvilke muligheter programmet gir.

1.2 Formål

Lage et datasystem som kan håndtere boligformidling i utleiemarkedet. Utleiere skal få sine tilbud presentert og ha mulighet til å nå potensielle leietakere. Boligsøkende skal kunne registrere sine opplysninger og sende inn ønsket bolig som finnes tilgjengelig for utleie. Firmaet skal registrere alle leieforhold som opprettes slik at boligen og leietakeren ikke lenger vil være registrert som henholdsvis ledig boligsøkende.

1.3 Tolkning av oppgaven

For å kunne gjennomføre prosjektoppgaven på den tid som vi har tilgjengelig har vi valgt og gjøre noen tolkninger og tilpasninger. I de følgende avsnitt følger en beskrivelse av hvordan vi har tolket oppgaven. Oppgaven gir oss stor frihet til å prioritere hva som skal implementeres, og noen egenskaper andre ville implementert er kanskje ikke blitt det i vårt prosjekt, og omvendt. Dette har vi likevel prøvd å legge til rette for ved en fremtidig utvidelse av programmet. Et eksempel på dette er statistikk. Vi har laget `Calendar`-felter i `Annonse`-objektet som er tenkt brukt i forbindelse med hvor lenge en annonse har vært annonsert, hvor mange dager det tok før den ble utleid osv. Vi har ikke fått implementert en god nok statistikkmodul som tar i bruk dette grunnet tidsmangel.

Vår prioritering har i hovedsak vært rettet mot å lage et så robust program som mulig fra bunnen av, slik at man enkelt kan bytte ut/bygge om store deler av programmet uten at andre funksjoner og funksjonalitet skal merke det. Mye av tiden vi har brukt på utvikling har dermed gått til å finne de beste strukturelle løsningene, selv om det har vært mer tidkrevende og tilsynelatende ikke vises i det kjørbare programmet. Dokumentasjonen vil derfor fokusere mye på de ulike løsningene vi har valgt og hvorfor vi har gjort det slik.

1.3.1 Brukergrensesnitt

Et slik program burde normalt bli delt opp i to separate programmer eller grensesnitt, slik at man fullstendig adskiller grensesnittet for megler og boligsøker. Med tanke på at data fra programmet skal serialiseres til disk ved avslutning av programmet skulle slik løsning medføre store utfordringer da to uavhengige programmer deler på samme register. For å enkelt demonstrere mulighetene har vi derfor valgt å sette opp begge deler av programmet inn i et og samme brukergrensesnitt. Grensesnittet for megler og utleier er derfor delt opp i samme vindu med `JTabpanes`.

1.3.2 Boligøskere

Til forskjell fra det som står i oppgaven har vi valgt å ikke registrere opplysninger til en boligsøker før den bestemmer seg for å melde sin interesse for en bolig/annonse. Boligsøker skal i sitt brukergrensesnitt bli presentert for alle tilgjengelige boligannonser filtrert utfra egne kriterier. Dersom man ønsker å søke på en bolig vil søkeren bli presentert med et dialogvindu for å legge inn sine opplysninger og sende dem til ansvarlig megler. Før søkeren kan registrere sine opplysninger vil han bli presentert med eventuelle krav fra utleier gjeldende den boligen (f.eks at det ikke er lov til å røyke i boligen). Søkeren må akseptere kravene for å kunne sende en forespørsel. En boligsøker kan søke på flere annonser, og leietakerobjektet blir lagt med som parameter i søknadsobjektet, men ikke registrert i personregisteret før en søknad er godkjent av megler. Denne boligsøkeren vil da ikke kunne søke på flere boliger, da leietakerobjektet er opprettet i personregisteret.

1.3.3 Meglere

I oppgaveteksten står det: «...Ved å matche ledige boliger mot de boligsøkendens beskrivelser skal firmaet informere sine kunder om aktuelle leietakere og boliger...»

Som beskrevet i 1.3.2 har vi valgt å legge slik matching av boliger på selve søkeren. Hvis boligsøker aksepterer utleiers krav under søkeprosessen vil det være bindende forhold i kontrakten også. Slike tilpasninger gjør at megleren stort sett har følgende ansvarsområder som kan håndteres via programmet: (1) registrering av nye utleiere, (2) registrering av boliger som tilhører utleierne, (3) legge ut annonser slik at de blir tilgjengelige for boligsøkere, (4) håndtere innkomne forespørsler, (5) opprette kontrakt mellom søkere og utleiere ved inngått avtale. Megler har da utleiers fullmakt til å håndtere leieforholdet på vegne av utleier. Vi har tolket det slik at megleren har tilgang til følgende registre:

- Søknader
- Annonser
- Utleiere
- Leietakere
- Kontrakter

1.3.4 Historikk

Dersom en leietaker inngår en kontrakt blir `Leietakerobjektet` sendt med som parameter, sammen med `Boligobjektet` og `Meglerobjektet` til et `Kontraktobjekt`¹. I kontraktregisteret skal det ikke være mulig for megler å foreta sletting slik at data over hvilke boliger ble utleide når til hvilke leietakere. Dette vil da utgjøre historikken som vil være søkbar, selv etter at en eventuelt bolig eller leietaker ikke lenger finnes i andre registre.

1.3.5 Utleiere

En utleier kan ha en eller flere boliger men kan også være en representant for et firma. En utleier kan be om å bli slettet fra registeret, men ikke om han har boliger i boligregisteret. Alle boliger til eieren må bli slettet først.

1.3.6 Boliger

Vi går ut fra at boliger som registreres til utleie er eiendomsobjekter som kan være tilgjengelige på utleiemarkedet så snart et leieforhold er over (dersom slik funksjonalitet ønskes av utleieren). Derfor har vi valgt å legge til funksjonaliteten i boligregisteret å sette opp dato fra hvilket en bolig kan være tilgjengelig for en ny uteie. Eventuelle bilder som blir lagret for boligen følger derfor boligobjektet og ikke annonsen slik at megleren ikke trenger å legge ut

¹f.eks. `Kontrakt kontrakt = new Kontrakt(Bolig, Megler, Leietaker);`

bildene på nytt dersom boligen skal plasseres på markedet igjen. En bolig som er utleid kan ikke slettes fra boligregisteret.

1.4 Mål

Følgende mål ble satt opp ved begynnelsen av arbeidet med oppgaven:

Robusthet Det var viktig at vår kode skulle være så generell så mulig slik at vi enkelt kan innføre eventuelle tillegg eller endringer i programmet med den hensikt at kodebasen blir enkel å vedlikeholde. Koden skal derfor bygges opp med hjelp av arv og polyformisme i så stor grad som mulig. Det skal være lite bruk av parametere eller identifikatorer som låser komponenter annet enn i konstanter som skal være låst.

MVC Programmet skal være bygget opp etter MVC²-arkitektur slik at logikk og brukergrensesnitt er helt avskjermet fra hverandre og all informasjonsutveksling blir håndtert via en kontroller. Det skal være så mye gjenbruk av kode som mulig. De metoder og variabler som kan være `static` skal være det.

Intuitivt brukergrensesnitt Brukergrensesnittet skal være enkelt og oversiktlig slik at en bruker som ikke er kjent med programmet kan foreta boligsøk og sende forespørsel til meglerfirmaet. En ny megler skal raskt starte opp i sin modul og på kort tid kunne bli kjent med programmets funksjonalitet.

Faglig utfordring Det var et mål at vi strakk oss langt i forhold til å komme opp med løsninger som ikke bare løser oppgaven i henhold til pensum, men på en måte som er mest mulig slik vi tror at man ville gjort det i næringslivet. Det vil si å ikke ta snarveier, velge `JTable` foran `JList`, bruke MVC, osv.

1.5 Tekniske detaljer

1.5.1 Utviklingsmiljø

Prosjektet er utviklet i NetBeans og Eclipse IDE³. Ikoner og annen grafikk er opprettet eller editert i Gimp⁴. Generelle ikoner (Open source) er hentet fra flaticons.net. Innledende struktur over klasser ble opprettet som UML diagram med ArgoUML. Hele prosjektet er lagd i tegnoppsett UTF-8 og det er ikke brukt noen norske bokstaver i kode eller kommentarer.

1.5.2 Krav til programvare

Programmet er kompilert med `javac 1.7.0_51` og testet på tilsvarende java versjon (`OpenJDK Runtime Environment`). Programmet er grundig testet på Linux (Ubuntu 12.04 64-bit, Fedora

²eng. Model View Controller

³eng. Integrated Development Environment

⁴The GNU Image Manipulation Program

Core 20 64-bit) samt Mac OS X. Det er gjennomført tilsvarende standard funksjonstester på MS Windows 7 og 8.1 (64-bit) for å verifisere plattformuavhengighet.

1.5.3 Versjonshåndtering

Til versjonhåndtering brukte vi GIT via terminal og innebygd støtte i utviklingmiljøer (IDE). Lagring av prosjektet ble gjennomført sentralt via en repository på github. Repository for gruppen er privat frem til innlevering av prosjektoppgaven og kommer til å gjøres tilgjengelig for publikum etter at deadline for prosjektet har utløpt. Kildekoden og prosjektets historikk vil da være tilgjengelig fra følgende linker:

Kildekode

<https://github.com/CervecerosCodigo/ServiciosDeVivienda>
Lagret som NetBeans Java SE prosjekt.

Rapport

<https://github.com/CervecerosCodigo/ServiciosDeViviendaInforme>
L^AT_EX kode

Kapittel 2

Prosessdokumentasjon

Dette kapitlet beskriver kort om prosessen fra prosjektet startet og tiden frem mot innlevering. Det er ikke brukt mye tid på hvem som har gjort hva, og til hvilken tid. Vi har heller delt det opp slik at vi skriver om hvorfor vi har gjort som vi har gjort og hvilken utfordringer vi støtte på underveis.

Både prosessdokumentasjonen og produktdokumentasjonen er full av referanser til de kapitlene og seksjonene der man kan lese mer om de forskjellige løsningene, og kodeksempler på hvordan det er løst.

2.1 UML

For å visualisere samspill mellom grunnleggende klasser og datafelt i programmert ble det tegnet opp et UML-diagram over grunnleggende register klasser (se vedlegg A figur A.1, side 82). Ut fra UML diagrammet ble det generert kodeskjelett til disse klassene, som utgjorde starten på prosjektet. Følgende klasser (og tilhørende private datafelt) er satt opp i UML:

- *abstract Person*
 - Megler
 - Utleier
 - Leietaker
- *abstract Bolig*
 - Enebolig
 - Leilighet
- Annonse
- Søknad
- Kontrakt

2.2 Arkitektur

Programmets arkitektur bygger på *Model View Controller* filosofi. Dette medfører at alle data og grafiske brukergrensesnitt er helt separert fra hverandre i egne moduler og pakker. All kommunikasjon mellom disse håndteres gjennom dedikerte **kontrollere** og **interface**. Denne arkitekturen ble valgt tidlig i prosjektet med tanke på å holde koden så robust som mulig. En slik tilnærming ga gjorde at de klassene som har med brukergrensesnittet å gjøre bare har den type komponenter, og ikke noe annet. Alt en trenger i brukergrensesnittet er **get-metoder** som gjør at kontrollene kan kommunisere med komponentene.

All logikk som håndterer funksjoner til grafisk grensesnitt for et enkelt vindu finnes i en egen klasse. Dersom MVC ikke hadde vært brukt i prosjektet hadde det resultert i meget store klasser, mest sannsynlig som består av flere tusen rader per vindu i noen tilfeller. MVC gav oss mulighet til å raskt legge til nye funksjoner til en allerede eksisterende vinduklasse. Et godt eksempel på dette er dersom samme vindu skal brukes både for registrering og editering av samme opplysninger. MVC-løsningen stiler kun krav om at kontrolleren har to konstruktører, én som setter opp vinduet dersom det skal foretas en nyregistrering og én konstruktør som kan brukes for endringer. Denne vil da opprette vindu og hente inn all nødvendig data fra objektet (model).

Løsningen vi har kommet frem til har ytterligere gjenbruk av kode da det er samme de kontrollere som ligger bak både «meglersarkfanen» og «annonsearkfanen». I det man oppretter kontrollene, sender man med hvilket vindu kontrolleren gjelder for. Man vil dermed ha én fysisk java-fil som styrer logikken til to vinduer som har tilnærmet lik funksjonalitet.

2.3 Arv og polymorfisme

I prosjektet ble det benyttet arv og polymorfisme i stor utstrekning med tanke på å gjenbruke så mye kode som mulig. Dette følte vi var krav for å lage et produkt med robust kode der alle moduler er separert fra hverandre. Samtidig har man veldig god kontroll på hva som må endres eller utbedres ved videre utvikling. Mye av komponentene vi bruker har utspring i én, eller veldig få superklasser, og ved endring av disse kan vi forandre store deler av programmet på en enkel måte.

Både personobjekter og boligobjekter har flere subklasser. Det samme har tabellens tabellmodell. Her brukes **instanceof** for å sjekke hvilken type objekter man har med å gjøre.

2.3.1 Swing komponenter

Prinsippet for arv er i størst utstrekning brukt ved GUI-komponenter. De komponentene som ble oftest gjenbrukt ble redefinert i form av abstrakte klasser eller laget som «Custom»-komponenter. Eksempel på dette er **JPanel**, **TextField** eller **Button**. Ved å la disse custom-komponentene arve de opprinnelige komponentene fra **Swing** kunne vi implementere en custom-komponent i stedet, og dermed raskt raskt endre alle de tilsvarende komponentene utseende og funksjon fra en sentral plass i programstrukturen, isteden for «*refactoring*»

av komponentens definisjon over programmets alle filer. Omtrent ingen komponenter brukes direkte fra **Swing** uten først å ha blitt tilpasset for gjenbruk flere steder.

2.3.2 Pakkefordeling

Programmet ble grundig delt opp i forskjellige pakker med tanke på å gruppere tilhørende komponenter sammen, og adskille dem fra komponenter som ikke logisk sett hører sammen. Dette er gjort spesielt med tanke MVC arkitektur. For eksempel alle klasser som bygger opp vinduer er plassert i **view**-pakke, deretter alle klasser som bygger opp registrerings vinduer er plassert **view.register**-pakken. For å gjenspeile denne analogien på controller-siden har vi lagt dem respektivt i **controller**-pakken og **controller.register**-pakken. Deretter ble alle klasser som **Person** eller **Bolig** lagt i pakken **model**. For grundig beskrivelse av fordeling av pakker og filer, se produktdokumentasjon avsnitt 3.1 side 17.

2.3.3 Konstanter og enum

Konstanter ble brukt med tanke på å ha en sentral definisjon av data som feks. konfigurasjoner. En sentral plassering av konstanter som **Regex** eller dimensjoner for **swing**-komponenter gir mulighet til å sette og hente definisjoner til komponentene vi bruker fra en sentral plass. Enum-typer brukes spesifikt for å sette faste definisjoner angående et spesielt objekt. Et godt eksempel på dette er *sivilstatus* der vi fyller opp en comboboks med enum-verdier i stedet for strenger.

2.4 Arbeidsfordeling

Tabell 2.1: Arbeidsfordeling mellom gruppemedlemmer

Utført av	Espen	Lukas	Petter
Dokumentasjon	• • •	• • •	•
UML	•	• • •	•
Main kontroller	• • •	•	•
Testing og testrapport	• • •		
GUI - struktur	• • •	• •	•
GUI Megler	•	• •	• • •
GUI Boligsøker		• • •	•
Tabellfunksjonalitet	• • •		
Utskrift(HTML av objektene)	• • •		
Serialisering	•	• • •	
Søkefunksjon		• • •	
Filterfunksjon		• • •	
Menyer (høyreklikk)	• • •		
Kontrollere registrering og endring	•	• • •	•
GUI registrering og endring	•	• • •	•
Innloggingsfunksjonalitet			• • •
Tastatursnarveier osv			• • •
Bilder visning og lagring		• • •	
Ikoner		• • •	•
GIT - oppsett og vedlikehold	•	• • •	

2.5 Utfordringer

Under veis i prosjektet har vi støtt på mange utfordringer. Hvis vi skal skrive ned alle så kan det resultere i at listen blir lengre enn selve rapporten. Vi har derfor valgt til å avgrense oss til de mest frekvente og de som gav oss størst utfordringer.

2.5.1 Kontrollerhierarki

I MVC-modellen, kan kommunikasjonen mellom kontroller og brukergrensesnitt håndteres av en metode i GUI-klassen som tar i mot en lytter som en parameter fra kontrolleren. Situasjonen blir litt annerledes dersom man bruker kontrollere som ligger parallelt eller i en hierarki med hverandre uten å ha en naturlig måte for dem å kommunisere med hverandre. Den utfordringen har vi løst ved bruk av lytter-interfacesom implementeres av alle disse klassene. Dersom lytter-interface «går av» vil det bli oppfattet av alle som som har implementert interfacet. Utfordringen med den løsningen at koden blir fort ganske kompleks men vinsten blir god fleksibilitet.

2.5.2 Serialisering

De utfordringer som vi støtte på ved serialisering av objektene var at i begynnelsen ble det prøvd løsninger der vi serialiserte registrene i en klasse spesifikk laget for dette. Med det menes at vi lagde en klasse som kunde ta imot våre **collections** (dataregistre) via sin konstruktør og deretter serialisere dem. Data ble serialisert som tenkt men vi fikk ikke til å lese inn dataene igjen for å gjenopprette objektene på nytt. Den eneste løsningen som fungerte var at vi serialiserte alle registre (**HashSet**) og static variabler i samme klasse der de ble instansiert. Dette er en vel fungerende løsning men er ikke noe som vil være robust nok dersom man ønsker å øke programmets funksjonalitet.

2.5.3 Tabellmodell

Det vi begynte på prosjektet hadde vi ikke lært mye om **JTable**. Da dette dukket opp på forelesning hadde vi allerede brukt det en stund, og funnet ut at vanlig **Array** ikke holder mål. Oracle sin dokumentasjon nevner bruk av **Array** og **Vectorer** som datakilde, men vi valgte å gå over til **ArrayList**. Vi har ett objekt per linje, og trengte ikke å bruke multi-dimensjonel implementasjon. Første forsøk på å lage egen tabellmodell gikk tilsynelatende bra, men ved fjerning av elementer fra tabellen oppstod det enormt mange **Null Pointer**- og **ArrayOutOfBounds**-exceptions. Spesielt om tabellen ble tømt for objekter, da hang hele programmet seg. Det tok en hel dag å bli kvitt dette problemet.

Første forsøk på å lage egen tabellmodell gikk tilsynelatende bra, men ved fjerning av elementer fra tabellen oppstod det enormt mange **Null Pointer**- og **ArrayOutOfBounds**-exceptions. Spesielt om tabellen ble tømt for objekter. Da hang hele programmet seg. Det tok en hel dag å bli kvitt dette problemet.

2.5.4 Layout-managere

Under prosjektets gang har vi prøvd alle **Layout**-managere i **AWT** klassen. Vi har hatt mange forskjellige behov underveis, i de forskjellige registreringsvindue og panelene, og har endt opp med en god løsning der vi stort sett bruker **BorderLayout** på selve vinduet, og så en variasjon av **FlowLayout** og **GridBagLayout** innvendig i panelene til **BorderLayout**. I de vinduene vi bare har ett panel har vi brukt **GridLayout** med én rad og én kolonne.

Alle våre paneler arver enten *AbstractPanel.java* eller er instans av *CustomSubPanel.java* (som igjen arver *AbstractPanel.java* og begge de klassene tar i mot **Layout**-managere i minst én av sine konstruktører.

2.5.5 Konflikter i GIT

Ingen i gruppen hadde vært borte i **GIT** noe spesielt før prosjektet startet og spesielt de første to ukene brukte vi like mye tid på å lære oss dette og fikse konflikter, som vi brukte på å programmere. Da to av oss bruker **Linux** og **Netbeans** og én **Mac** med **Eclipse**, så har det vært en del utfordringer knyttet til dette også. Men å ta i bruk et versjonhåndteringssystem som

GIT har vært en god læring og gitt oss ny kunnskap som vi kan ta med oss videre til andre prosjekt under utdannelsen og ut i arbeidslivet.

2.5.6 Objektreferanser

I klassen `FreeTextSearch.java` har vi støtt på problemer med testing av objekttype som skulle brukes i klassens søkemetoder. Ettersom vi bruker arv av `Person` for de forskjellige instansene av person som `Utleier`, `Megler` eller `Leietaker` ønsker man at det skal være mulig å teste på hvilken type av person som blir hentet fra registeret. For søking bruker vi generiske metoder, disse kan håndtere alle mulige register av type `HashSet`, ikke bare de som består av type `Person`. Ettersom vi vet hvilken type av set som vi kommer til å sende inn i metoden så hadde det vært optimalt å sende med en parameter som beskriver hvilken type av person som det skal søkes på. Se eksempel 2.1, rad 6.

Eksemplet viser en mulighet på hvordan vi ønsker at dette bør fungere. Som siste parameter sender vi inn en referanse til det objekt som vi skal teste på. I den indre if-testen forsøker vi å hente opp navn til objektets instans or å undersøke objektets type. Dette er testet på mange forskjellige måter og kombinasjoner uten noe som helst resultat. Det er også testet med enum type som spesifiserer alle grunnleggende objekttyper som inngår i programmet (`Objekttype2.java`). Det var dessverre ikke mulig å bruke enum-typer for å undersøke objektinstansen via `instanceof`. Den testen i eksempelet er åpenbart feil, og om man velger teste på «`Utleier`» så går det selvfølgelig fint, men da blir den ikke så generisk som ønskelig. Ønsket var å ha en metode som fungerer uansett hva man tester på, uten å måtte spesifisere manuelt hva den skal testes mot.

Eksempel 2.1: Problem med å teste for objekttype (rad 6).

```
1      public ArrayList<T> searchForPatternIUtleier(HashSet<? extends Searchable>
2          liste, String pattern, Object obj) {
3          pattern = pattern.trim();
4          pattern = pattern.toLowerCase();
5          ...
6          for (Searchable o : liste) {
7              if (o instanceof obj.getClass()) {
8                  checkMeForResults = o.toSearch();
9                  for (String s : checkMeForResults) {
10                     s = s.toLowerCase();
11                     if (s.contains(pattern))
12                         resultList.add((T) o);
13                 }
14             }
15             ...
16         }
```

2.5.7 Swing og portabilitet

Det har vært noen utfordringer med diverse `Swing`-komponenter. Vi opplevde til stadighet at tekstfelt «kollapset». Det vil si at de ble bare ca 1cm bred. Neste gang man åpnet samme vindu kunne det være i orden igjen.

En annen ting er `JEditorPane`. HTML-versjonen som støttes er versjon 3.2, og CSS 1.0. Det er ikke mulig å plassere komponenter slik man er vant til i dag, og bruk av tabeller ble eneste løsning. Dette er likevel noe vi gjerne skulle ha gjort bedre dersom teknologien bak `JEditorPane` hadde tillatt det. `JavaFX` har støtte for HTML5.

`JPopupMenu` som vi bruker ved høyreklikking i tabellen var svært ustabil da vi brukte Java's innebygde "Cross platform Look and Feel" kunne popupmenyen dukke opp et god stykke unna der man høyreklikket. Dette var variabelt fra gang til gang man åpnet programmet. Etter overgang til Nimbus Look and Feel har dette, samt flere andre ting blitt betraktelig bedre.

Et problem som vi fortsatt har i Linux, men som virker å være borte i Windows, er at hver gang man høyreklikker i tabellen så må vi gjøre det to ganger for at den skal skifte fokus fra det den holdt på med. Hadde man feks hatt personobjekter i tabellen, og så søker på nytt etter boliger, så kunne man likevel få opp menyen for personobjekter ved det høyreklikket. Høyreklikket man en gang til dukket boligobjektenes meny opp.

2.5.8 Genereiske metoder og klasser

Det har også oppkommet noen utfordringer gjeldene generiske typer, metoder og klasser. Det er generelt ganske vanskelig å sette seg inn i programmering med generiske typer. Selv om kan kjenner til prinsippet og hvordan det skal brukes i programmet så er utfordringen mest relatert til selve syntaksen for generiske variabler. Dette bli spesifikk vanskelig dersom det skal brukes diamantoperator sammen med eventuelle wildcards. Oftest må det til ganske mye testing i form av «try and fail» før man får frem noe som fungerer på den måte som man har tenkt seg. Under prosjektets gang har det inntruffet flere ganger at man har gitt opp en god løsning som bygger på generiske typer på grunn av at vi ikke klarte å sette sammen en kode som ble godtatt av kompilatoren, selv om det i teorien burde ha fungert.

Kapittel 3

Produktdokumentasjon

3.1 Pakker

Da implementasjonen vi har lagt oss på krever mange javafiler, som ofte gjerne har relativt lite innhold har vi også delt javaklassene inn i flere pakker under mappen “src”.

controller

Controller og undermappen Registrer inneholder all funksjonalitet til alle JPanels, deres innhold og registreringsvinduer. Funksjonaliteten og nærmere beskrivelse av disse klassene finnes i andre deler av dokumentasjonen.

lib

Lib er biblioteket av statiske metoder, konstanter og enum, samt andre klasser og metoder som ikke naturlig hører til MVC-tankegangen. Her finnes også våres RegEx-konstanter som er brukt på alle Tekstfelt i GUI, samt på nytt i kontrollerne. Det testes altså to ganger, før registrering og validering før objektet opprettes/endres.

model og register

Model inneholder klassene det lages objekter av, som igjen legges inn i datasettene. Klassene Person og Bolig er abstrakte med hver sine subklasser. Klassen TabellModell er en abstrakt klasse som arver DefaultTableModel, og den har igjen klasser som arver dens funksjonalitet. Dette vil en kunne lese mer om i beskrivelsen av tabellimplementasjonen. Pakken Register inneholder tomme klasser. Hensikten med å ha egne registerklasser, ut over det å ha alle HashSet i MainController er å implementere serializable. Det er altså disse klassene som serialiseres, og som inneholder alle datasett som blir skrevet til og lest fra fil.

search

Search inneholder filene for søkeimplementasjonen vi har utviklet. Dette er en helt egenutviklet løsning som vil bli beskrevet nærmere i avsnittet for søkeimplementasjonen 3.8, side 43.

serviciosdevivienda

Det er to klasser her. Mainmetoden vår, `serviciosdevivienda.java`, samt `SkrivTilLesFraFil.java` som utfører all serialisering av data. Serialisering blir dokumentert nærmere her 3.3, side 19.

view og undermappen registrer

Disse to pakkene inneholder alle brukergrensesnitt, egendefinerte javakomponenter som *CustomJTextField*, *CustomSubPanel* osv. Det er brukt en stor mengde kreativitet og tankevirk-somhet for å komme frem til løsningen vi har endt opp med, og det vil blir beskrevet flere andre steder i rapporten.

3.2 Start og avslutting

Programstart foretas fra pakke `ServiciosDeVivienda` fra en main metode med samme navn. Programmet blir startet med bruk av `SwingUtilities` hvilket gjør det mulig at `Swing` komponentene blir startet i en «trygg» programtråd som er tilpasset for bruk med `Swing` klassen (se eksempel 3.1). Med slik tilnærmingsmåte forventes det at GUI delen av programmet skal bli mer stabilt.

Programmet blir også avsluttet med hjelp av `ShutdownHook` på en måte som er anbefalt for JVM. Løsningen gir en mulighet til å avslutte programmet på en «naturlig» måte slik at eventuelle alle konkurrerende tråder får kjøre parallelt frem til avslutting. Løsningen sikrer at data som blir serialisert og skrevet til fil med mindre sjanse for feil og eventuelle skrive og lese feil.

Eksempel 3.1: Oppstart av programmet.

```
1      //Start
2      SwingUtilities.invokeLater(new Runnable() {
3
4          SkrivTilLesFraFil startProgram;
5
6          @Override
7          public void run() {
8              startProgram = new SkrivTilLesFraFil();
9              //Avslutting
10             Runtime.getRuntime().addShutdownHook(new Thread(new Runnable()
11                 {
12                     @Override
13                     public void run() {
14                         System.out.println("Programmet avsluttes");
15                         startProgram.lagreData();
16                     }
17                 }
18             ));
19         }
20     });
```

```
15         }
16     }));
17 }
18 }));
```

3.3 Serialisering

Skriving og innlesing av data blir kalt opp fra programmets `main` metode. Les og skriv metoder finnes i `SkrivTilLesFraFil.java`. Den klassen initierer `MainController.java` som er programmet primære kontroller i MVC arkitekturen. Alle registre blir initialisert fra les/skrive klassen med hensikt å ha mulighet til å seriasliere alle registrene på plass. Eksempel 3.2 viser hvordan registrene og static variabler blir håndtert ved skrivning og serialisering. Programets arkitektur baserer seg på tellevariabler som holder kontroll på antall objekter som til hver gang blir opprettet og lagt til i hvert enkelt register. Statiske filer blir skrevet i spesifikk rekkefølge etter at alle registrene (`HashSet`) er skrevet til fil. Ved innlesing av lagret data blir alle registrene passert som komponenter til `MainKontroller`, se eksempel 3.3.

Lesing og skriving er opprettet etter Java SE 6 struktur, derfor benytter vi oss ikke av «try-med resurser» tilnærming. Hensikt med den brukte løsningen er at det oppleves som at vi får en mer oversiktlig kode.

Eksempel 3.2: Serialisering og skriving av data.

```
1 public void lagreData() {
2     try {
3         FileOutputStream fos = new FileOutputStream(new File(Konstanter.
4             FILNANV));
5         ObjectOutputStream out = new ObjectOutputStream(fos);
6
7         out.writeObject(personliste);
8         out.writeObject(boligliste);
9         out.writeObject(annonseliste);
10        out.writeObject(kontraktliste);
11        out.writeObject(soknadsliste);
12
13        out.writeInt(Person.getTeller());
14        out.writeInt(Bolig.getTeller());
15        out.writeInt(Annonse.getTeller());
16        out.writeInt(Soknad.getTeller());
17        out.writeInt(Kontrakt.getTeller());
18
19        out.close();
20    } catch (IOException e) {
21        System.out.println(e.fillInStackTrace());
22    }
23 }
```

Eksempel 3.3: Innlesing av serialisert data.

```
1 public void lesInnData() {
2     try {
```



```

3      FileInputStream fis = new FileInputStream(new File(Konstanter.
        FILNANV));
4      ObjectInputStream in = new ObjectInputStream(fis);
5
6      personliste = (HashSet<Person>) in.readObject();
7      boligliste = (HashSet<Bolig>) in.readObject();
8      annonseliste = (HashSet<Annonse>) in.readObject();
9      kontraktliste = (HashSet<Kontrakt>) in.readObject();
10     soknadsliste = (HashSet<Soknad>) in.readObject();
11
12     Person.setTeller(in.readInt());
13     Bolig.setTeller(in.readInt());
14     Annonse.setTeller(in.readInt());
15     Soknad.setTeller(in.readInt());
16     Kontrakt.setTeller(in.readInt());
17
18     in.close();
19     mainController = new MainController(personliste, boligliste,
        annonseliste, kontraktliste, soknadsliste);
20
21     } catch (IOException e) {
22         System.out.println(e.fillInStackTrace());
23
24     } catch (ClassNotFoundException e) {
25         System.out.println(e.fillInStackTrace());
26     }
27 }

```

3.4 MVC

Programmeringskonseptet Model View Controller er noe vi så vidt har vært gjennom i pensum, men om helt klart var noe av det første vi bestemte oss for å ta i bruk. Et hvert dataprogram står i fare for å ende opp med enormt store GUI-klasser om man ikke gjør noen grep. I tillegg vil all kode være eksklusiv for den klassen og ikke kunne gjenbrukes fra andre klasser. Har man to-tre vinduer som skal utføre omtrent samme type oppgaver ender man opp med å skrive omtrent identisk kode for både brukergrensesnittet og funksjonaliteten flere ganger. Man står også i fare for veldig omstendelig kode som er vanskelig å vedlikeholde, og som gjør videre utvikling og utvidelse av funksjonalitet betraktelig vanskeligere.

3.4.1 Generelt om den strukturelle oppbyggingen

Da boligsøkere har behov for å søke etter boliger trenger de både et søkepanel, en tabell eller listevissning av søkerresultatene og en visning av de enkelt annonseobjektene. Vi la også merke til at megler ville ha omtrent det samme behovet, bare med mulighet for å behandle boliger, personer, søknader og kontrakter i tillegg.

Derfor har vi valgt å løse ved å lage to "versjoner" av samme brukergrensesnitt og kontroller-funksjonalitet. Det vil si en **Layout** basert på **BorderLayout** med et søkepanel på toppen, en tabell på venstreside, et resultatvindu i sentrum og et knappespanel i bunnen. Vi har dermed endt opp med en ramme av **toppPanel**, **venstrePanel**, **senterPanel** og **bunnPanel** som i

sin tur tar inn paneler i form av egne klasser. På denne måten har vi kunnet lage to versjoner av `toppPanel`, én for `meglerVindu` og én for `annonseVindu`. De andre panelene er felles for begge visningene, men kunne selvsagt byttes ut eller deles opp på tilsvarende måte relativt enkelt og raskt.

I de følgende delkapitlene vil det gås nærmere inn på hvordan de forskjellige programmeringslagene henger sammen og jobber seg i mellom.

3.4.2 Oppstart av kontrollere og brukergrensesnitt

Den første klassen som instansieres fra `SkrivtilLesFraFil.java` under oppstart er `MainController.java`. Det er fra denne kontrolleren alt annet starter, og det er her programmet «deler» seg i tre. Denne kontrolleren har da det overordnede ansvar for å holde programmet sammen, og har da tilgang til alle relevante kontrollere, brukergrensesnitt og datasett. Ut over å starte opp brukergrensesnitt og kontrollere finnes det noen implementasjoner av spesialsydde «lyttere». Da alle våre paneler er egne javaklasser og søkeknappen i `toppPanel` skal returnere et datasett til `venstrePanel` sin tabell, så har vi laget `interface` som kontrolleres fra `MainController.java`. Skjer det en endring i `toppPanel`, nærmere bestemt i `ActionPerformed`-metoden, så sendes søkeresultatet via `MainController.java` til `venstrePanel` sine tabell-metoder. Dette helt uten at kontrollere for de to panelene vet om hverandre. Bruken av slike lyttere er brukt flere steder i programmet og vil bli dekket, med eksempler her 3.7.12 på side 42.

Når alle kontrollere er instansiert og brukergrensesnittet opprettet så kjøres det to metoder som setter opp flere lyttere i `ControllerTabell.java`, samt sende et datasett til hvert av de vinduenes tabeller.

3.4.3 Oversikt over hvilken kontrollere som hører til hvilken GUI-klasse

Tabell 3.1: Oversikt over kontrollerne og deres tilhørende GUI-klasser

Kontroller	GUI-klasse
ControllerBildeViser.java	BildeViser.java
ControllerBunnPanel.java	BunnPanel.java
ControllerKeyBindings.java	
ControllerOutput.java	SenterPanel.java
ControllerTabell.java	VenstrePanel.java
ControllerToppPanelMegler.java	TopPanelMegler.java
ControllerToppPanelAnnonse.java	TopPanelAnnonse.java
Innloggingskontrollerer.java	LoggInnDialog.java
MainController.java	
ControllerRegistrerAnnonse.java	AnnonseRegVindu.java
ControllerRegistrerBolig.java	BoligRegVindu.java
ControllerRegistrerLeietaker.java	PersonRegVindu.java
ControllerRegistrerSoknad.java	SoknadRegVindu.java
ControllerRegistrerUtleier.java	PersonRegVindu.java

3.5 Datastruktur (Model)

3.5.1 Valg av datastruktur

Alle dataregistreene er laget på `HashSet` fra `Collection`-rammeverket til Java. Vi har brukt `HashSet` for å unngå dobbellagring av like objekter. Vi har «overridet» `equals` og `hashCode`-metodene for objektene slik at de bare relevante datafelter brukes for å identifisere hva som er et unikt objekt og ikke. For eksempel så er det autoinkrementerte ID-feltet som gir alle objekter en unik ID, utelatt fra `equals` og `hashCode` da alle objekter da ville bli oppfattet som unike objekter. Videre har vi ikke hatt bruk for sortering av dataene våre. Alle datasett blir sortert når de vises i tabellen.

I pakken `register` ligger selve registerne som igjen inneholder hver sitt `HashSet`. Alle registerklassene arver `Register.java` som har en del generiske metoder som fungerer på alle `HashSet`ene og deres objekter.

3.5.2 Datatyper

Alle objektene, `Person`, `Bolig`, `Annonse`, `Søknad`, og `Kontrakt` har hver sin statiske teller som tildeles hvert objekt som opprettes. `Bolig`-objektene har teller fra 20000 til 29999. `Annonse` fra 30000 til 39999 osv. Alle objektklassene implementerer interfacet `Searchable` som vi har utviklet selv. dette interfacet har en metode som heter `toString` som returnerer et array av `String`-elementer. Denne metoden definerer hva som skal inngå i fritekstsøket som beskrives mer her 3.8 på side 43.

Bolig

Klassen `Bolig.java` er abstrakt og har to subklasser; `Leilighet` og `Enebolig`. Grunnet begrensning i tid har vi ikke valgt å implementere flere typer boliger enn dette. Boligobjektene har også en variabel `erUtleid`. Om en bolig er utleid vil ikke den kunne slettes fra datasettet `boligliste`. Det er også en variabel som definerer en filsti til bilder av boligen. Informasjon om denne funksjonaliteten kan man lese mer om her 3.9.

Person

Klassen `Person.java` er abstrakt og har som `Bolig.java` også subklasser; `Utleier.java`, `Leietaker.java` og `Megler.java`. Det er ikke implementert all funksjonalitet man gjerne skulle ønske seg på disse klassene. Megler-klassen har ikke noe registreringsvindu. Det er hardkodet en megler for å kunne vise funksjonaliteten med pålogging til `meglerVindu`, men det er lagt større vekt på å få `Leietaker` og `Utleier` godt implementert da det er disse som skal behandles i annonser, søknader osv.

Annonse

Annonseobjektet består i hovedsak av et `Bolig`-objekt pluss noen egne felter for utleierpris, depositum, utleiers krav til leietaker osv. Det er her også noen `Calendar`-variable som er tiltenkt bruk ved fremtidig utvidelse av programmet. Metoder `datoAnnonseRegistrert` og `datoAnnonseTasAvNett` vil kunne gi megler veldig mange muligheter for å lage statistikk på hvor lenger en annonse har vært på nett før den leies ut osv.

Søknad

En søknad består av et annonseobjekt, leietakerobjekt, søknadsID og to boolske variabler for om søknaden er behandlet og godkjent. Det er lagt opp til at mange søknader på samme annonse kan havne i megler sin liste over innkomne søknader. Søknadene sorteres da først synkende på kolonnen `erBehandlet`, og så stigende på kolonnen `annonseID`. I det én søknad blir godkjent av megler så vil alle andre søknader automatisk bli markert som behandlet og som "ikke godkjent". Det er foreløpig ikke implementert en tilbakemeldingsfunksjon til boligsøkerne om deres søknader er godkjent eller avslått, men det ligger det selvfølgelig til rette for.

Kontrakt

En Kontrakt opprettes automatisk i det en søknad godkjennes. Annonseobjektet, Meglerobjektet og Leietakerobjektet, samt info om leiepris, depositum og dato kontrakten ble inngått er hoveddelene i kontrakten, og denne informasjonen kan ikke slettes eller endres. Optimalt sett skulle vi hatt en utskriftsvennlig versjon av en slik kontrakt, men det er ikke funnet tid til dette.

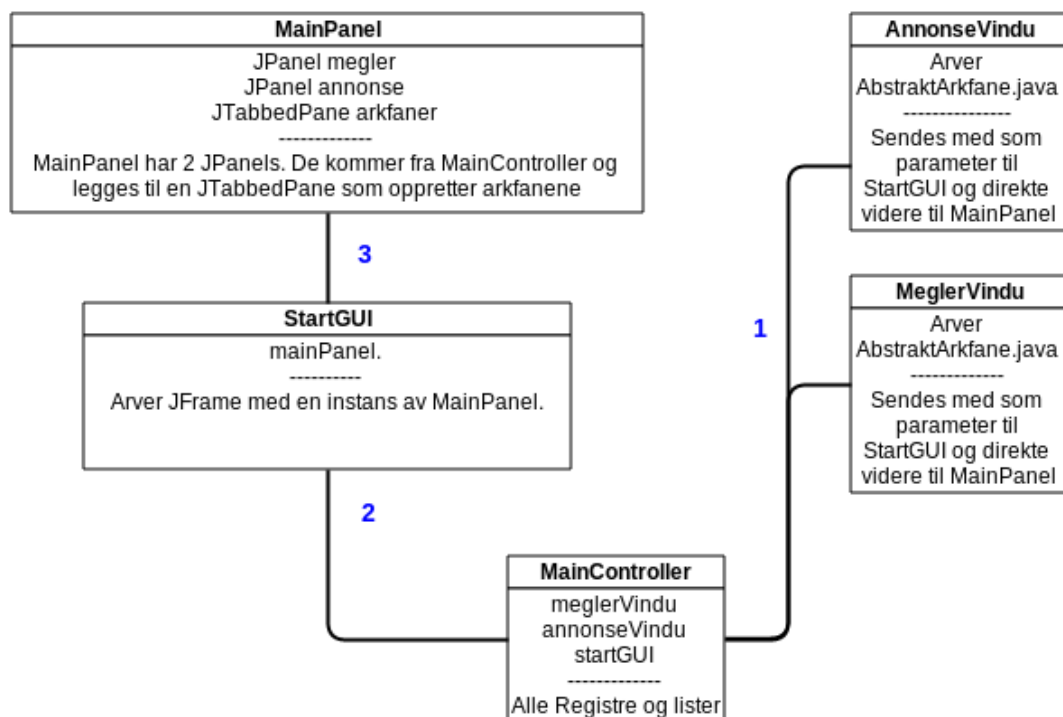
3.6 Brukergrensesnitt (View)

Dette delkapittelet er litt overlappende med kapittelet "Swing-komponenter". Les gjerne mer om det her 3.11 55.

3.6.1 Oppstart av brukergrensesnittet

I `MainController.java` instansieres først to arkfaner, `ArkfaneMegler.java` og `ArkfaneAnnonse.java`. Disse sendes så med som parametere inn i strukturen til brukergrensesnittet, først til `StartGUI.java`, og så videre til `MainPanel.java` som oppretter `JTabbedPane` og legger inn de to arkfanene der. Grunnen til at arkfanene ble opprettet allerede i `MainController.java` er fordi vi sender med det respektive arkfane under instansiering av kontrollerne. Siden de to arkfanene har så lik funksjonalitet, deler de på kontrollerne som de har felles.

Fra `MainController.java` opprettes det to versjoner av `ControllerTabell.java`. Til den ene sendes vinduet `meglerVindu` som er instans av `ArkfaneMegler.java` og til den andre kontrollereinstansen sendes `annonseVindu` som er instans av `ArkfaneAnnonse.java`. Se figur ??.



Figur 3.1: Illustrasjon over hvordan MainController starter opp programmet
labelfig:maincont

3.6.2 Bruk av arv i brukergrensesnittet

Bruker grensesnitt er utrolig tidskrevende å jobbe med, mye av det man gjør er å skrive samme kode for hvert enkelt vindu. For å unngå dette mest mulig, samt å få alle vinduer og paneler til å ha mest mulig den samme følelsen har vi noen abstrakte klasser som vi arver på tvers av alle klassene som har med brukergrensesnitt å gjøre.

AbstractPanel.java

Dette panelet arver `JPanel` og er superklassen til alle andre hovedvinduer og paneler i programmet, klassen inneholder to konstruktører. Den ene tar inn to heltall som representerer bredde og høyde på det panelet som arver klassen, samt en `String` `borderTitle` som blir tittel på kantlinjen som konstruktøren oppretter. Den andre konstruktøren er omtrent lik, men tar ikke inn en `String` for kantlinje, så her kan man velge mellom to konstruktører, med eller uten kantlinje. Dette panelet er også satt med en bakgrunnsfarge slik at alle paneler og vinduer som arver denne vil arve denne.

`CustomSubPanel.java` arver også `AbstractPanel.java`. Denne klassen brukes i hovedsak for paneler inne i et hovedpanel. Da dette panelet skal brukes gjentatte ganger i mange forskjellige sammenhenger er det hele 6 konstruktører her. I noen opprettes det en predefinert `Layout`, i andre ikke. Noen tar inn størrelse (bredde og høyde), andre ikke. På denne måten har vi fått en veldig slagkraftig klasse som har spart oss for mye arbeid og forenklet konstruksjonen av brukergrensesnittet veldig.

AbstractRegistreringsVindu.java, AbstractRegistreringsPanel.java

`AbstractRegistreringsVindu.java` er superklassen for `AbstractRegistreringsPanel.java`. Denne klassen arver `JFrame` og er setter de mest generelle parameterne som et hvert vindu skal ha. Størrelse, navn på vinduet, samt standard lukkefunksjonalitet osv. `AbstractRegistreringsPanel.java` er superklassen til alle registreringsvinduene. Denne klassen har en `BorderLayout` og tar inn parametere som bredde, høyde og tittel for vinduet som opprettes. Ut over dette gjør ikke denne klassen mye. Alle klasser som arver denne vil velge hvilke paneler fra denne klassens `BorderLayout` de ønsker å ta i bruk.

3.6.3 Oppbyggingen av arkfanene

Begge arkfanene er bygget over samme lest. De arver `AbstraktArkfane.java`. Denne klassen tar inn en `String` som avgjør hvilke `topPanel` som skal følge med arkfanen. `AbstraktArkfane.java` består i korte trekk av en `BorderLayout` med fire paneler; `toppanel`, `venstrepanel`, `senterpanel` og `bunnpanel`. Hvert av panelene er satt opp med en `get`-metode som brukes hyppig av kontrollerne i kommunikasjonen med komponentene der.

Alle kontrollerne er «klar» over sitt vindu, og nedenfor er et eksempel (3.4) på hvordan `ControllerTabell.java` kommuniserer med vinduet.

Eksempel 3.4: ControllerTabell.java kommunikasjon med brukergrensesnitt.

```

1      /**
2      * Lytter på museklikk i Output-vinduet.
3      */
4      vindu.getSenterpanel().getEditorPane().addMouseListener(new
        MouseAdapter() {
5
6          @Override
7          public void mouseClicked(MouseEvent e) {
8              if (e.getButton() == MouseEvent.BUTTON1) {
9                  if (modellIBruk instanceof TabellModellAnnonse) {
10                     Annonse valgtObjekt = returnerAnnonseObjekt();
11                     if (vindu instanceof ArkfaneMegler) {
12                         new ControllerBildeViser(valgtObjekt.getBolig(),
13                             true);
14                     } else {
15                         new ControllerBildeViser(valgtObjekt.getBolig(),
16                             false);
17                     }
18                 }
19             }
20         });

```

I figuren ovenfor kan man både få tak i vindu sine paneler, men også teste om vindu er av type `ArkfaneMegler` eller ikke. Tidligere i prosjektet brukte vi **konstanter** som ble sendt med som parametere for å identifisere hvilke vindu man var i, og hvilken objekttype man behandlet til en hver tid.

TopPanelMegler.java, TopPanelAnnonse.java, VenstrePanel.java, SenterPanel.java og BunnPanel.java

De tre nevnte panelene utgjør innholdet i hver av arkfanene. Da all funksjonalitet finnes i de respektive **kontrollerne** er hver av disse klassene ganske små. De består i grunn bare av en **Layout** samt den/de komponentene som vises, inkludert subpaneler samt **get-metoder** en trenger for å kunne hente informasjon derfra. Det er brukt forskjellige **Layout-managere**, alt etter hvilken som er mest hensiktsmessig i den gitte situasjon. I **VenstraPanel.java** og **SenterPanel.java** har man bare én komponent hver som dominerer hele panelet, men **kontrollerne** bak disse er desto større. **JTable** i **VenstrePanel.java** har utrolig mye funksjonalitet og er hjerte i hele programmet. **JEditorPane** i **SenterPanel.java** er en ren **output-pane** som skriver ut html-visning av objektene i tabellen.

3.6.4 Registreringsvindue

Registreringsvindue er bygget over samme lest, og arver **AbstractRegistreringsPanel**. De er likevel ganske forskjellige. De bruker et utall forskjellige **Layout-managere**, og ofte en blanding mellom flere. **GridBagLayout** er allikevel den mest allsidige og foretrukne **Layout-manageren** vi har brukt.

Så og si alle komponentene vi har brukt er `Custom...`. Det vil si egendefinerte komponenter vi har laget, og som arver den opprinnelige `Swing`-komponenten. Det beste eksempelet her er `CustomJTextField.java` som blant annet tar inn et `String pattern` som er en `Regex`-konstant, slik at komponenten har innebygget `Regex`-funksjonalitet.

Alle registreringsvinduerne har i sine kontrollere egne konstruktører ment for nye objektet og for endring av objektene. Det vil si at når en endrer et objekt så fylles komponentene ut med relevant informasjon fra det innkomne objektet.

Når en velger ny `Bolig` eller har valgt et valg som gjelder bare for én type objekter så vil de valgene som ikke gjelder for det valget bli deaktivert og ikke mulig å legge inn informasjon på. Dette er selvfølgelig funksjonalitet som ligger i `kontrollerne` og vil bli nevnt i 3.7.9 på side 38.

Vi har gjort et eksperiment med `PersonRegVindu.java`. Dette er et felles registreringsvindu for både `Leietakere` og for `Utleiere`. Vinduet har to `kontrollere` som kaller på vinduet ved behov. Dette betyr at en del komponenter vil være overflødige i noen tilfeller, og nødvendige i andre tilfeller. Dette vil bli nevnt mer i 3.7.9 på side 38.

3.7 Kontroller (Controller)

3.7.1 Generelt om kontrollermiljøet

Da prosjektet startet var vi fornøyd med én `kontroller` og kalte den `MainController.java`. Det viste seg ganske raskt at den ville kunne bli enormt stor, så da endte vi opp med å dele den opp i flere deler. Siden den gang har vi besluttet at vi like godt kunne ha én `kontroller` til hvert vindu, og kanskje noen uten tilhørende vinduer også. `MainController` instansierer alle `hovedkontrollerne`, bortsett fra `BunnController` som instansieres fra `ControllerTabell` og `ControllerOutput` som ikke instansieres. Den har bare statiske metoder som tar i mot diverse data og viser dem i `JEditorPane` formatert som `HTML`.

I tillegg til `hovedkontrollerne` har vi `registreringskontrollerne` som hører til hvert sitt registreringsvindu. De vil bli nevnt under 3.7.9 på side 38.

Det er et eget avsnitt i dette kapitlet vedrørende lyttedfunksjonaliteten vi har implementert for å kunne kommunisere mellom de forskjellige `kontrollerne`.

I den første delen av prosjektet var ikke `JTable`-implementasjonen vår veldig god og vi kunne ikke benytte oss av å teste hvilket datasett som var satt i tabellen. Dermed endte vi opp med å bruke identifikatorer for dette. Først med `konstanter` og så `Enum` som identifiserte personobjekter, boligobjektet osv. Vi måtte derfor implementere `Interface`-lyttere som lyttet på når en ny `ArrayList` var satt i tabellen blant annet.

Siden det har tabellen kommet skikkelig på plass og det går nå an å teste hvilken type objekter som ligger i tabellen. Dermed har vi til dels begynt å teste på tabellens `getModel()`-metode for å finne instansen av datasettet.

Siden vi likevel er veldig stolte av løsningen vår med `interface`-lyttere har vi valgt å beholde noe av implementasjonen og vil derfor ha et eget delkapittel for dette her 3.7.12, på side 42.

3.7.2 Hovedkontrollerne

Hovedkontrollerne er betegnelsen på klassene:

- `MainController.java`
- `ControllerTabell.java`
- `ControllerOutput.java`
- `ControllerBunnPanel.java`
- `ControllerToppPanelMegler.java`
- `ControllerToppPanelAnnonse.java`

Disse kontrollerne styrer absolutt all funksjonalitet mellom datastrukturen og brukergrensesnittet.

3.7.3 `MainController.java`

Denne klassen instansieres fra `SkrivTilLesFraFil.java`, der data blir lest inn fra fil og skrevet til fil. Denne klassen har ikke noe brukergrensesnitt og den eneste oppgaven den har er å starte andre kontrollere og brukergrensesnittet. I tillegg til det passer kontrolleren på at de andre komponentene som skal kommunisere sammen har mulighet til det. Dette er løst med en lytteløsning beskrevet her 3.7.12, på side 42.

3.7.4 `ControllerToppPanelMegler.java`

Brukergrensesnittet til denne klassen gir megler mulighet til å søke i de forskjellige datasettene, samt registrere nye objekter. I tillegg er det en enkelt statistikkpanel som viser antall ledige boliger og antall kontrakter signert i år.

Klassens primære funksjon er søkefunksjonaliteten og videresending av resultatet til tabellen. Metoden `finnValgtObjektITabell` har en `ListSelectionListener` som lytter på tabellens `valueChanged`. Hver gang en ny rad er valgt i tabellen så returneres nummeret på raden som er valgt. Om verdien som returneres er forskjellig fra `-1` (når tabellen ikke har en valgt rad returneres `-1`) så konverteres nummeret til den bakenforliggende tabellmodellens index. Til slutt så lagres det et `Object valgtObjekt` som hentes på den indexen som er angitt.

Metoden `sendSokeResultat` tar i mot en streng fra søkefeltet, og avhengig av hvilken `radioknapp` som er valgt så kjøres søkemetoden på det rette datasettet. Resultatet lagres i en `ArrayList`. Søkemetoden er beskrevet nærmere her 3.8, på side 43. Resultatet blir så sendt ned til tabellen og vist der. Det skjer via `lytteren` som blir vist ved eksempel her 3.7.12, på side 42. Den private klassen `KnappeLytter` lytter på de fire knappene i brukergrensesnittet. Metodene tester på hvilken type objektet som er valgt i tabellen er og man får bare trykke på vedrørende knapp om det man ønsker å utføre er relevant. Eksempelvis så skal man ikke kunne trykke på `Ny Bolig` uten å ha først valgt et personobjekt av typen `Utleier` i tabellen.

3.7.5 ControllerToppPanelAnnonse.java

Denne kontrolleren er mer avansert enn `ControllerToppPanelMegler.java`. Klassen benytter seg av et `AnnonseFilter` som er beskrevet i detalj her 3.8.2, på side 45. `Annonsefilteret` har to `get`-metoder som brukes for å fyllet ut de to komboboksene som lister boligtyper og poststeder. Det er metodene `addPoststederToComboBoxOnLoad` og `addBoligTyperToComboBoxOnLoad` som tar seg av denne jobben. Komboboksen som lister opp poststed er implementert slik at den søker gjennom alle annonsene som er publisert og henter poststedet inn i listen. Det vil si at når én annonse forsvinner, så forsvinner også poststedet fra komboboksen også, så sant det ikke er andre annonser publisert for en bolig på samme poststed.

I det man trykker på søkeknappen i brukergrensesnittet så kjøres metoden `sendSokeResultat`. Denne metoden kjører så metoden `filtrerAnnonser`, og om lytteren er instansiert så sendes datasettet ned til tabellen via `interfacet` som står for kontakten mellom kontrollerne. Denne lyttefunksjonaliteten blir vist ved eksempel her 3.7.12, på side 42.

Metoden `filtrerAnnonser` nevnt overnfor henter søkekriterier fra brukergrensesnittet og om kriteriene passerer RegEx-testen så sendes de med til `annonsefilteret` som parametere. Returen fra denne metoden er et `HashSet` av annonser.

3.7.6 ControllerBunnPanel.java

Denne klassen instansieres fra `ControllerTabell.java` i motsetning til de andre kontrollerne. Denne klassen har tre knapper som brukes i forbindelse med navigering i tabellen, samt til å åpne registreringsvinduet for valgt objekt for endring. Klassen har én privat `KnappeLytter`-klasse som utgjør omtrent hele funksjonaliteten til denne kontrolleren. Den ene knappen her har forskjellig funksjonalitet om man er i `annonseVindu` enn om man er i `meglerVindu`. Konstruktøren til `KnappeLytteren` endrer navn på `MultiKnapp` som vi har valgt å kalle den. I `actionPerformed`-metoden vil det testes for hvilket datasett som ligger i tabellen, hvilken rad som er valgt, og hvilket vindu man befinner seg i. Se følgende eksempel 3.5:

Eksempel 3.5: Utdrag fra `actionPerformed`-metoden i `ControllerBunnPanel.java`

```
1  if (tabell.getModel() instanceof TabellModellAnnonse) {
2      modellIBruk = (TabellModellAnnonse) modellIBruk;
3      if (vindu instanceof ArkfaneMegler) {
4          new ControllerRegistrerAnnonse(annonseliste, personliste, (Annonse
5              ) modellIBruk.finnObjektIModell(valgtRad));
6      } else {
7          new ControllerRegistrerSoknad(personliste, annonseliste,
8              soknadliste, (Annonse) modellIBruk.finnObjektIModell(valgtRad)
9              );
10     }
```

3.7.7 ControllerOutput.java

`SenterPanel.java` har én komponent som viser valgt objekt i tabellen i HTMLvisning. Alle metodene i denne klassen er statiske. De skal ikke behandle objekter, bare vise dem. Denne

kontrolleren er derfor ikke instansiert noe sted. Alle «HTML-metodene» tar i mot `JEditorPane`-komponenten det skal skrives til, i hvilket vindu man befinner seg i, objektet som skal vises og eventuelt et eller flere `HashSet` om en trenger ytterligere informasjon i utskriften.

`Utleier`-objekter har i visningen blant annet en liste over boliger de eier, om noen. Da er det hjelpemetoder som itererer over `boligregisteret` og finner eieren av hver bolig. Boligene til valgt valgt `Utleier` vil da returneres og skrives ut i `Utleier`-objektet.

Ut over selve metodene for å vise objekter så finnes metoden `setStylesheet` helt nederst i klassen. Alle «HTML-metodene» må formateres så de passer inn i vinduet `JEditorPane` har tilgjengelig. Det er stort sett brukt tabeller for å vise objekter, og da `JEditorPane` ikke støtter nyere HTML-versjon enn 3.2 med CSS 1.0, så har vi vært ganske begrenset på hvor for seg gjort visningene kunne bli. Det måtte en del justering på plass, da en ikke kan automatisk skalere bilder, som i nyere versjoner av HTML.

3.7.8 ControllerTabell.java

Denne kontrolleren er holder orden på alle data som skal vises i tabellen i `VenstrePanel.java` der tabellen ligger, samt funksjonalitet for å slette, endre og opprette nye objekter. Dette kapittelet vil ta for seg noen nøkkelpunkter på funksjonalitet og flyt av data.

- Hvordan tabellen tar i mot data og settes opp, side 30
- Datamodellene, sortering og formatering, side 32
- `PopupMenu`-funksjonalitet, side 35
- Lytter på klikk i `Output`-vinduet, side 35

Hvordan kontrolleren setter opp tabellen

I det programmet starter opp kaller `MainController.java` opp to metoder i `TabellController.java`, `settOppTabellLyttere` og `settInnDataITabell`. Den første av disse tar for seg en hel del initialisering av tabellens virkemåte. Den andre tar i mot et datasett og setter det inn i tabellen. Den metoden vil bli beskrevet mer i avsnittet for tabellens virkemåte 3.7.8, side 32.

En `JTable` har muligheter for utrolig mye funksjonalitet om en velger å ta det i bruk. Vår implementasjon er for så vidt litt annerledes enn en del av eksemplene en kan lese om andre steder. Vi har valgt å ikke tillate endring i tabellen. En `JList` fant vi fort ut var for enkel, da vi gjerne også ville kunne sortere på kolonner.

I prosessen har vi endret tabellens datastruktur to ganger. Først ble det blant annet brukt en `array`, men da den er vanskelig å slette fra uten mye ekstra prosessering så endte vi opp med en `ArrayList` som datastruktur. Da vår tabell skal liste ett objekt per linje, så valgte vi ikke å bruke en flerdimensjonal `array`.

Metoden `settOppTabellLyttere` gjør litt mer enn bare å sette opp lyttere. Først initialiserer den `ControllerBunnPanel.java` sin private `lytteklasse`. Videre kalles metoden `settOpplyttereForPopupMenuITabell` som definerer funksjonaliteten til `PopupMenu` når en høyreklikker i tabellen. Les mer om den på 3.7.8, på side 35. Metoden kobler så tabellen

og popupmenyen sammen via tabellens `setComponentPopupMenu`-metode, før den initialiserer slettefunksjonalitet i tabellen når en bruker Delete-knappen på tastaturet. Se følgende eksempel (3.6):

Eksempel 3.6: Slettefunksjonalitet i tabellen ved å trykke Delete på tastaturet.

```

1      inputMap = tabell.getInputMap(JTable.WHEN_FOCUSED);
2      actionMap = tabell.getActionMap();
3
4      Action sletteknappFunksjon = new AbstractAction() {
5          @Override
6          public void actionPerformed(ActionEvent e) {
7              generellSletteMetodeSomKallerOppRettSletteMetode();
8          }
9      };
10
11     inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_DELETE, 0), "Slett");
12     inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_BACK_SPACE, 0), "Slett");
13     actionMap.put("Slett", sletteknappFunksjon);

```

Metoden som kalles opp `generellSletteMetodeSomKallerOppRettSletteMetode` kaller igjen på den rette metoden som faktisk utfører slettingen. Det er tre slettemetoder i kontrolleren; `slettPerson`, `slettBolig` og `slettAnnonse`.

Metoden er nå kommet for å initialisere diverse lyttere. De vil vises med eksempler. Først ut er tabellens funksjonalitet for å oppfatte hvilken rad som er valgt, og deretter kalle på metoden `sendObjektFraTabellTilOutput`. Det vil si at hver gang en ny linje velges i tabellen så kjøres denne `sendObjektFraTabellTilOutput`-metoden, som plukker opp hvilken type objekt som er valgt og sende det til output-vinduet. Se følgende eksempel (3.7):

Eksempel 3.7: Lytteren som finner valgt rad i tabellen.

```

1      tabell.getSelectionModel().addListSelectionListener(new
2          ListSelectionListener() {
3
4          @Override
5          public void valueChanged(ListSelectionEvent e) {
6
7              if (e.getValueIsAdjusting()) {
8                  return;
9              }
10             try {
11                 int rad = tabell.getSelectedRow();
12                 if (rad > -1) {
13                     rad = tabell.convertRowIndexToModel(rad);
14
15                     //Lagrer raden i en variabel, som brukes i andre
16                     metoder.
17                     valgtRadItabell = rad;
18                     sendObjektFraTabellTilOutput(objekttype);
19                 }
20             } catch (ArrayIndexOutOfBoundsException aiobe) {
21                 System.out.println("Tabell ConvertRowIndexToModel
22                     ArrayIndexOutOfBoundsException");
23             } catch (IndexOutOfBoundsException iobe) {

```

```

22         System.out.println("Tabell ConvertRowIndexToModel
23             IndexOutOfBounds");
24     }
25 }

```

De resterende lytterne vil bli dekket i avsnittet om interaktivitet i tabellen 3.7.8, på side 35.

Tabellens oppsett, virkemåte og formatering

En `JTable` kan implementeres på forskjellige måter. Ofte holder det med å implementere en `DefaultTableModel`, men i vårt tilfelle der vi har forskjellige typer objekter som krever forskjellige kolonnenavn så har vi valgt å la klassen `TabellModell.java` arve `DefaultTableModel` og override de metodene vi har behov for. Videre så vil subclassene til `TabellModell.java`, som `TabellModellAnnonse.java` spesifisere hvilke kolonnenavn man har i det datasettet, samt hvilke felter i et annonseobjekt som skal hentes til hvilken kolonne.

Klassen `VenstrePanel.java` hvor tabellen instansieres er også der man finner noe av tabellens utvidete funksjonalitet.

Metoden `settCelleRenderer` definerer hvordan spesifikke celler i tabellen skal formateres. Her er det bare definert én regel og det er for behandlede `Søknadsobjekter`. Metoden finner ut hvilken `TabellModell` som er i bruk, og så tester den kolonnen «Er behandlet» (nummer 2) i `TabellModellAnnonse.java` om søknaden er behandlet eller ikke. er den behandlet skal cellen dimmesned. Metoden gjengis i sin helhet i eksempel 3.8 under:

Eksempel 3.8: Metoden `settCelleRenderer` fra `VenstrePanel.java`.

```

1  public void settCelleRenderer() {
2
3      tabell.setDefaultRenderer(Object.class, new DefaultTableCellRenderer()
4          {
5              @Override
6              public Component getTableCellRendererComponent(JTable tabell,
7                  Object verdi, boolean erValgt, boolean harFokus, int rad, int
8                  kolonne) {
9                  TabellModell modell = (TabellModell) tabell.getModel();
10
11                  Component c = super.getTableCellRendererComponent(tabell,
12                      verdi, erValgt, harFokus, rad, kolonne);
13                  if (modell instanceof TabellModellSoknad) {
14                      if (tabell.getValueAt(rad, 2).equals("Ja")) {
15                          c.setForeground(new Color(200, 200, 200));
16                      } else {
17                          c.setForeground(Color.BLACK);
18                      }
19                  }
20                  c.repaint();
21                  return c;
22              }
23          });
24 }

```

Metoden `resizeKolonneBredde` kalles opp hver gang et nytt datasett er lagt inn i tabellens modell. Metoden består av to `for`-løkker inne i hverandre som itererer gjennom alle cellene i tabellen. For hver kolonne så finner metoden den cellen som bruker mest plass"og setter kolonnebredden til det. Metoden er gjengitt i sin helhet i eksempel 3.9 under:

Eksempel 3.9: Metoden `resizeKolonneBredde` fra klassen `VenstrePanel.java`

```

1  public void resizeKolonneBredde() {
2      TableColumnModel kolonneModell = tabell.getColumnModel();
3      Component comp = null;
4      TableCellRenderer renderer = null;
5
6      for (int kol = 0; kol < tabell.getColumnCount(); kol++) {
7          int bredde = 50; //minste bredde
8          for (int rad = 0; rad < tabell.getRowCount(); rad++) {
9              renderer = tabell.getCellRenderer(rad, kol);
10             comp = tabell.prepareRenderer(renderer, rad, kol);
11             bredde = Math.max(comp.getPreferredSize().width, bredde);
12         }
13         kolonneModell.getColumnModel().setPreferredWidth(bredde);
14     }
15 }
```

Metoden `sorterTabellVedOppstart` definerer en sorteringsrekkefølge på tabellens første kolonne, slik at tabellen alltid er sortert på ID-feltet til gjeldene objekt. Metoden `sorterTabellSoknadData` gjelder bare for `Søknadobjekter` i `TabellModellSoknad.java`. Denne sorterer først synkende på kolonnen "Er behandlet", og deretter på ID-kolonnen. Man vil da alltid ha ubehandlede søknader liggende øverst i tabellen, sortert innad på `AnnonseID`.

Hvilken av disse sorteringsmetodene som tas i bruk bestemmes metoden `settInnDataITabell` i `ControllerTabell.java`. Metoden tar inn et datasett i form av `HashSet` eller `ArrayList`, samt en `Enum`-variabel som identifiserer hvilken `radioknapp` som er valgt i søkepanelet i enten `TopPanelMegler.java` eller `TopPanelAnnonse.java`. Metoden har en `switch/case` på `Enum`-variabelen og utfører så de instruksjoner som er gjeldende for det datasettet som skal inn i tabellen. Hele metoden er gjengitt i eksempel 3.10:

Eksempel 3.10: Metoden `settInnDataITabell` i `ControllerTabell.java`

```

1  public void settInnDataITabell(Collection innkommendeDatasett, ObjektType
    objektTypeEnum) {
2
3      if (innkommendeDatasett.size() > 0) {
4          tabellData = new ArrayList<>();
5          Iterator<?> iter = innkommendeDatasett.iterator();
6          while (iter.hasNext()) {
7              tabellData.add(iter.next());
8          }
9
10         try {
11             switch (objektTypeEnum) {
12                 case PERSONOBJ:
13                     this.objekttype = ObjektType.PERSONOBJ;
14                     tabellModellPerson.fyllTabellMedInnhold(tabellData);
15                     tabell.setModel(tabellModellPerson);
16                     tabellModellPerson.fireTableStructureChanged();
17                     modellIBruk = tabellModellPerson;
```

```

18         vindu.getVenstrepanel().sorterTabellVedOppstart();
19         break;
20     case BOLIGOBJ:
21         this.objekttype = ObjektType.BOLIGOBJ;
22         tabellModellBolig.fyllTabellMedInnhold(tabellData);
23         tabell.setModel(tabellModellBolig);
24         tabellModellBolig.fireTableStructureChanged();
25         modellIBruk = tabellModellBolig;
26         vindu.getVenstrepanel().sorterTabellVedOppstart();
27         break;
28     case ANNONSEOBJ:
29         this.objekttype = ObjektType.ANNONSEOBJ;
30         tabellModellAnnonse.fyllTabellMedInnhold(tabellData);
31         tabell.setModel(tabellModellAnnonse);
32         tabellModellAnnonse.fireTableStructureChanged();
33         tabell.getColumnModel().getColumn(2).setCellRenderer(
34             vindu.getVenstrepanel().
35                 settHoyrestilltFormateringPaaTabell());
36         tabell.getColumnModel().getColumn(3).setCellRenderer(
37             vindu.getVenstrepanel().
38                 settHoyrestilltFormateringPaaTabell());
39         modellIBruk = tabellModellAnnonse;
40         vindu.getVenstrepanel().sorterTabellVedOppstart();
41         break;
42     case KONTRAKTOBJ:
43         this.objekttype = ObjektType.KONTRAKTOBJ;
44         tabellModellKontrakt.fyllTabellMedInnhold(tabellData);
45         tabell.setModel(tabellModellKontrakt);
46         tabellModellKontrakt.fireTableStructureChanged();
47         modellIBruk = tabellModellKontrakt;
48         vindu.getVenstrepanel().sorterTabellVedOppstart();
49         break;
50     case SOKNADSOBJ:
51         this.objekttype = ObjektType.SOKNADSOBJ;
52         tabellModellSoknad.fyllTabellMedInnhold(tabellData);
53         tabell.setModel(tabellModellSoknad);
54         tabellModellSoknad.fireTableStructureChanged();
55         modellIBruk = tabellModellSoknad;
56         vindu.getVenstrepanel().sorterTabellSoknadData();
57         break;
58     }
59     vindu.getVenstrepanel().resizeKolonneBredde();
60     vindu.getVenstrepanel().settCelleRenderer();
61     bunnController.settOppTabellData(modellIBruk);
62
63     } catch (ArrayIndexOutOfBoundsException aiobe) {
64         System.out.println("settInnDataITabell gir ArrayOutOfBounds
65             ved innlegging av nytt datasett");
66     } catch (NullPointerException npe) {
67         System.out.println("settInnDataITabell gir NullPointerException ved
68             innlegging av nytt datasett");
69     }
70 } //End Try/Catch
71
72 } //End If datasett > 0
73 } //End Metoden settInnDataITabell

```

Som man ser av metoden ovenfor så er det en del instruksjoner som gjøres for hvert datasett,

men også noen unike for det enkelte datasett. F.eks så ser man at **Søknadsobjektene** sorteres med den tidligere metoden for den typen objekter. Til slutt i metoden så kjøres **resizemetoden** og **formateringsmetoden**.

Muselyttere og interaktivitet i tabellen

De resterende **lytterne** er forskjellige «muse-lyttere», som responderer på enkeltklikk, dobbelklikk og høyreklikk i tabellen.

Lytteren som følger nå er laget for å gi ekstra funksjonalitet til annonsene. Det vil si at for annonser med flere bilder kan man klikke hvor som helst i output-vinduet og få opp en bildeviser og blå gjennom disse bildene. Denne bildefunksjonaliteten er beskrevet mer her 3.9, på side 47.

Lytteren differensierer på om en er i annonsevinduet eller i meglervinduet. Er man i annonsevinduet så åpnes bildeviseren med den konstruktøren som ikke gir mulighet for å slette eller endre bildene. Dette er dog tillatt i meglervinduet, se eksempel 3.11.

Eksempel 3.11: Lytter for museklikk i output-vinduet.

```
1      vindu.getSenterpanel().getEditorPane().addMouseListener(new
      MouseAdapter() {
2
3          @Override
4          public void mouseClicked(MouseEvent e) {
5              if (e.getButton() == MouseEvent.BUTTON1) {
6                  if (modellIBruk instanceof TabellModellAnnonse) {
7                      Annonse valgtObjekt = returnerAnnonseObjekt();
8                      if (vindu instanceof ArkfaneMegler) {
9                          new ControllerBildeViser(valgtObjekt.getBolig(),
                          true);
10                     } else {
11                         new ControllerBildeViser(valgtObjekt.getBolig(),
                          false);
12                     }
13                 }
14             }
15         }
16     });
```

Den følgende **lytteren** som settes opp er del av tabellen sin **MouseAdapter**-implementasjon på lik linje som den som følger etter denne (eksempel 3.12). I **mouseClicked**-hendelsen når en dobbelklikker på en rad i tabellen vil starte opp registreringsvinduet for det valgte objektet og fylle ut all informasjon om objektet ut fra det som allerede er registrert. Så kan man endre objektet og trykk OK for å oppdatere objektet. Eksempelen nedenfor dekker bare **Bolig**-objekter. Det settes en **lytter** på **kontrolleren** til registreringsvinduet som er koblet til **actionPerformed**-metoden til vinduet. I det man trykker OK så vil metoden **oppdaterTabellEtterEndring** kalles og den den tilhørende **TabellModell** vil oppdateres. Det vil si at tabellens innhold vil oppdatere seg med den nye endringer. Det forutsetter at man har rett **TabellModell** valgt. Feks så kan en ikke stå i **TabellModellPerson** og så opp endringene for en ny bolig som ble lagt til. Tabellens **TabellModeller** vil diskuteres i avsnittet 3.7.8, på side 32.

Eksempel 3.12: Hendelse ved dobbelklikking på et objekt i tabellen.

```

1      @Override
2      public void mouseClicked(MouseEvent e) {
3
4          if (e.getClickCount() == 2) {
5
6              if (tabellModellBolig.equals((TabellModell) tabell.
7                  getModel())) {
8                  ControllerRegistrerBolig cont = new
9                      ControllerRegistrerBolig(boligliste,
10                         returnerBoligObjekt());
11                  cont.settTabellOppdateringsLytter(new
12                      TabellFireDataChangedInterface() {
13
14                          @Override
15                          public void oppdaterTabellEtterEndring() {
16                              tabellModellBolig.fireTableDataChanged();
17                          }
18                      });
19          }
20      }

```

Den siste lytteren i metoden `settOppTabellLyttere` er `mouseReleased`-funksjonen på høyremuseknapp. Det vil si når man høyreklikker i tabellen så vil en popupmeny dukke frem. De menyvalgene som dukker frem er avhengige av hvilket datasett man jobber på. Feks så kan man ikke høyreklikke på en annonse og så opp «Ny Bolig».

Det første utdraget (eks. 3.13) fra koden gjelder for hvilket menyvalg som er tilgjengelig når en høyreklikker på en boligobjekt og et personobjekt. Det andre utdraget (eks. 3.14) vil være fra metoden `settOpplyttereForTabellMenyITabell` som spesifiserer funksjonaliteten til selve menyvalgene i pop-up menyen.

Eksempel 3.13: Menyvalg ved høyreklikk i tabellen.

```

1
2      @Override
3      public void mouseReleased(MouseEvent e) {
4          if (e.getButton() == MouseEvent.BUTTON3) {
5
6              //Tømmer menyen før den tegnes på nytt.
7              tabellMeny.removeAll();
8
9              try {
10                 if (tabellModellBolig.equals((TabellModell) tabell.
11                     getModel())) {
12                     tabellMeny.add(menyvalgBolig);
13                     menyvalgBolig.add(menyvalgEndreBolig);
14                     menyvalgBolig.add(menyvalgSlettBolig);
15                     menyvalgBolig.add(menyvalgPubliserToggle);
16                 } else if (tabellModellPerson.equals((TabellModell)
17                     tabell.getModel())) {
18                     tabellMeny.add(menyvalgPerson);
19                     tabellMeny.add(menyvalgBolig);
20                     menyvalgPerson.add(menyvalgNyPerson);
21                     menyvalgPerson.add(menyvalgEndrePerson);
22                     menyvalgPerson.add(menyvalgSlettPerson);

```

```
22         if (returnerPersonObjekt() instanceof Utleier) {
23             menyvalgBolig.add(menyvalgNyBolig);
24         }
25
26     }
```

Eksempel 3.14: Funksjonaliteten til to av menyvalgene i pop-up menyen.

```
1  public void settOpplyttereForPopupMenyITabell() {
2
3      //Man ser bare dette valget om man høyreklikker på en utleier i
4      //tabellen
5      menyvalgNyBolig.addActionListener(new ActionListener() {
6          @Override
7          public void actionPerformed(ActionEvent e) {
8              Person valgtObjekt = returnerPersonObjekt();
9              if (valgtObjekt instanceof Utleier) {
10                 ControllerRegistrerBolig cont = new
11                 ControllerRegistrerBolig(boligliste, (Utleier)
12                 valgtObjekt);
13                 cont.settTabellOppdateringsLytter(new
14                 TabellFireDataChangedInterface() {
15                     @Override
16                     public void oppdaterTabellEtterEndring() {
17                         tabellModellPerson.fireTableDataChanged();
18                     }
19                 });
20             }
21         });
22
23         //Endre bolig
24         menyvalgEndreBolig.addActionListener(new ActionListener() {
25             @Override
26             public void actionPerformed(ActionEvent e) {
27                 Bolig bolig = returnerBoligObjekt();
28                 if (bolig != null) {
29                     ControllerRegistrerBolig cont = new
30                     ControllerRegistrerBolig(boligliste, bolig);
31                     cont.settTabellOppdateringsLytter(new
32                     TabellFireDataChangedInterface() {
33                         @Override
34                         public void oppdaterTabellEtterEndring() {
35                             tabellModellPerson.fireTableDataChanged();
36                         }
37                     });
38                 }
39             }
40         });
41     }
42 }
```

3.7.9 Kontrollerne for registreringsvinduerne

AbstractControllerRegister.java

Alle kontrollere for registrerings vinduer finnes i pakken `controller.register`. Samtlige av disse kontrollene arver `AbstractControllerRegister.java` (eksempel 3.15) som er superklasse til registerkontrollene. Superklassen brukes til å håndtere lesing og skrivning av objekter til alle `HashSet` som brukes i de kontrollene. Klassen gjør at koden for lesing og skriving blir arvet av hver enkel kontroller. Ettersom det brukes flere forskjellige `HashSet` i programmet er superklassen satt opp med generisk parametre og begrenset til `Collection.Set.HashSet<E>`. Superklassen består av to konstruktører, den første blir kalt opp med en mengde som parameter og brukes ved registrering av nytt objekt f.eks. ny bolig. Nødvendige parametre for opprettelse av objektet blir hentet opp fra `gui` gjennom en kontroller som arver den superklassen. Den andre konstruktøren tas i bruk dersom et boligobjekt skal editeres, derfor blir det sent med et set og et generelt objekt som blir sendt fra markeringen i tabellen. Slik løsning er valgt etterom `HashSet` ikke tillater duplikater, hvilket brukes som en automatisk mekanisme for å unngå dobbelregistrering av data i registrene. Dersom man skal gjennomføre endringer i et eksisterende objekt i settet, må det objektet plukkes ut fra mengde, slettes fra mengde, deretter kan datafelt oppdateres og legges tilbake i mengden. Derfor blir objektet som skal endres direkte sendt med til den andre konstruktøren i superklassen.

Eksempel 3.15: `AbstractControllerRegister.java` kontroller arvet av alle registreringskontrollerne.

```
1 public abstract class AbstractControllerRegister<E> {
2
3     final HashSet<E> set;
4     Object obj;
5
6     public AbstractControllerRegister(HashSet<E> set) {
7         this.set = set;
8     }
9
10    public AbstractControllerRegister(HashSet<E> set, Object obj) {
11        this.set = set;
12        this.obj = obj;
13    }
14
15    public boolean slettObjekt(E e) {
16        return set.remove(e);
17    }
18
19    public boolean registrerObjekt(E e) {
20        return set.add(e);
21    }
22 }
```

ControllerRegistrerBolig.java

Kontrolleren for boligvindu er satt opp med to konstruktører: en som sørger for registrering av en ny bolig, og den andre konstruktøren brukes til endring av eksisterende bolig (eksempel

3.16). Som nevnt i avsnitt 3.11.4 (side 57) er det ikke mulig å registrere en bolig uten eier. Den første konstruktøren tar derfor for seg to parametre: (1) HashSet over bolig objekter og (2) person instansen Utleier. De to viktigste oppgavene til denne konstruktøren er å starte opp vinduet til boligbehandling samt kalle opp konstruktøren til superklassen slik at tilhørende HashSet kan behandles av denne. Den andre konstruktøren som finnes med i klassen tar med set HashSet og et bolig objekt som det skal endres på. Datafelt til objektet blir endret med data som tas imot fra GUI. Hvis data som tas imot fra GUI går igjennom regex testen (se eksempel 3.17) kan objektet blir oppdatert, slettet fra HashSet og lagt det opdaterte obejktet vil bli lagt til registeret (HashSet) på nytt (eksempel 3.18).

Eksempel 3.16: ControllerRegistrerBolig.java: Oversikt over konstruktører i bolig registrering/endring kontrolleren.

```

1    /**
2     * En konstruktør for registrering av en ny bolig.
3     *
4     * @param boligSet HashSet<Bolig>
5     */
6    public ControllerRegistrerBolig(HashSet<Bolig> boligSet, Utleier utleier)
7    {
8        super(boligSet);
9        erNyregistrering = true;
10       boligBilde = new BoligBilde();
11       bRegVindu = new BoligRegVindu("Registrering av boliger");
12       bRegVindu.setKnappeLytter(new KnappeLytter());
13       bRegVindu.getEierField().setText(String.valueOf(utleier.getPersonID()));
14       bRegVindu.setIconImage(Ikoner.NY_BOLIG.getImage());
15       bRegVindu.deaktiverBildeKnapper();
16   }
17   /**
18    * En konstruktør som brukes for endring av en bolig.
19    *
20    * @param boligSet HashSet<Bolig>
21    * @param bolig Bolig
22    */
23   public ControllerRegistrerBolig(HashSet<Bolig> boligSet, Bolig bolig) {
24       super(boligSet, bolig);
25       erNyregistrering = false;
26       boligBilde = new BoligBilde();
27       this.bolig = bolig;
28
29       initialiseringAvController();
30
31       bRegVindu.setIconImage(Ikoner.EDIT.getImage());
32
33       //Sperrer mulighet til å gjøre om en leilighet til enebolig og vice
34       versa
35       bRegVindu.getLeilighetRButton().setEnabled(false);
36       bRegVindu.getEneboligRButton().setEnabled(false);
37   }

```

Eksempel 3.17: ControllerRegistrerBolig.java: Regex test av generelle tekstfelt for bolig.

```

1     private boolean kontrollerDataBolig() {
2
3         boolean[] boligOK = new boolean[7];
4
5         boligOK[0] = RegexTester.testID(String.valueOf(eierID));
6         boligOK[1] = RegexTester.testID(String.valueOf(meglerID));
7         boligOK[2] = RegexTester.testGateadresseEnkel(adresse);
8         boligOK[3] = RegexTester.testPostNummer(postNr);
9         boligOK[4] = RegexTester.testPostOrtNavn(postSted);
10        boligOK[5] = RegexTester.testKVMbolig(String.valueOf(boAreal));
11        boligOK[6] = RegexTester.testYearNummer(String.valueOf(byggeAr));
12        for (int i = 0; i < boligOK.length; i++) {
13            if (!boligOK[i]) {
14                return false;
15            }
16        }
17        return true;
18    }

```

Eksempel 3.18: ControllerRegistrerBolig.java: Sletting og oppdatering av samme bolig objekt via superklassen til kontrolleren.

```

1     private boolean slettBoligFraSet(Bolig bolig) {
2
3         if (kontrollerDataForSletting(bolig)) {
4             if (super.set.remove(bolig)) {
5                 return true;
6             }
7         }
8         visMelding("slettBoligFraSet", "Bolig ble IKKE slettet fra set");
9         return false;
10    }
11
12    private boolean skrivOppdateringTilBoligSet(Bolig bolig) {
13        if (super.set.add(bolig)) {
14            visMelding("skrivOppdateringTilBoligSet", "Boligen ble oppdatert i registret");
15            return true;
16        }
17        visMelding("skrivOppdateringTilBoligSet", "Boligen ble IKKE oppdatert i registret");
18
19        return false;
20    }

```

En viktig funksjon i alle kontrollene til registreringsvinduer er initialisering av lyttere i disse klasser. Eller registreingsvinduer har en metode som kan sette en `ActionListener` til vinduene som blir sendt fra en kontrollere klasse. Hver controllerklasse innholder en privat controller klasse `KnappeLytter` (implementerer `ActionListener`), se eksempel 3.19. Den private klassen blir alltid instasiert fra konstruktøren til registringskontrollere og sendt som parameter til vinduklassen (etter MVC arkitektur prinsipper).

Eksempel 3.19: ControllerRegistrerBolig.java: Uttidrag fra privat lytterklasse i kontrolleren.

```

1     private class KnappeLytter implements ActionListener {

```

```
2
3      @Override
4      public void actionPerformed(ActionEvent e) {
5          if (e.getSource().equals(bRegVindu.getLagreButton())) {
6
7              if (erNyregistrering) {
8
9                  ...
```

Pakken for registreringskontrollene består av totalt seks klasser:

- `AbstractControllerRegister.java`
- `ControllerRegistrerAnnonse.java`
- `ControllerRegistrerBolig.java`
- `ControllerRegistrerLeietaker.java`
- `ControllerRegistrerSoknad.java`
- `ControllerRegistrerUtleier.java`

I dette avsnittet beskrives kun `ControllerRegistrerBolig.java` men de andre klassene i denne pakken har mer eller mindre densamme funksjonalitet. Det som er den største forskjellen mellom disse er preliminær t forskjellige antall tekstfelt i vinduene og noen andre typer at datafelt (f.eks. `Calenderobjekt`). Ettersom funksjonaliteten og strukturen på de klassene er så lik den klassen som beskrives i dette avsnittet her kommer vi ikke til å beskrive disse i detalj. For de spesielt interesserte henviser vi derfor til kildekoden (pakke `controller.register`).

3.7.10 Innloggingskontroller

Denne kontrollern lytter på arkfanen `Megler` via en `ChangeListener`. Dersom nevnt arkfane blir valgt, så sjekkes det hvorvidt man allerede er innlogget eller ei via en static boolean variabel som er satt til `false` som standard. Dersom en ikke er innlogget, så vil man få mulighet til å logge inn via `JDialog LoggInnDialog.java`. Av enkelhets skyld, så har vi hardkodet en admin brukerkonto som kan brukes midlertidig, for å logge inn. Hjelpemetoden `sjekkAdmin()` eksisterer i sammenheng med dette.

Klassen mottar et `HashSet<Person>` fra `Maincontroller.java` via konstruktøren, som inneholder alle person objekter lagret på fil, inkludert eventuelt megler brukerkontoer. Innloggingsknappen tilhørende `LoggInnDialog.java`, lyttes på via en `ActionListener`. Dersom knappen trykkes på, så kalles metoden `sjekkInformasjon()` opp, i tillegg til nevnt hjelpemetode `sjekkAdmin()`. Førstnevnte metode itererer igjennom `HashSet`, og sjekker hvorvidt det finnes en megler med det aktuelle brukernavnet og passordet, som er skrevet inn i `LoggInnDialog` vinduet. Ved en match, så blir boolean variabelen `innlogget` satt til `true`, og brukeren vil bli videreført til `Megler` arkfanen. Dersom det ikke blir funnet en match, får brukeren beskjed om dette gjennom et `JOptionPane.showMessageDialog()` vindu. Ved å trykke på avbryt i innloggingsvinduet, blir brukeren brakt tilbake igjen til `Annonse` arkfanen.

3.7.11 Tastatursnarveier: ControllerKeyBindings

Denne kontrolleren mapper opp alle tastatursnarveier man kan bruke for å navigere i programmet. Via klassens konstruktør, så tas det imot de vindu objekter vi har behov for å jobbe med. Gjennom vinduene sine get-metoder, så kaller vi opp `getInputMap()`, og `getActionMap()` metodene tilhørende det aktuelle GUI objektet vi ønsker å legge til en tastatursnarvei til.

Som parametere i `getInputMap()` metoden, så brukes det en `JComponent` konstant for å sikre at den aktuelle tastatursnarveien kan brukes uavhengig av hvor i programvinduet brukeren har fokus. Dette i tillegg til det aktuelle `KeyStroke` objektet, samt en referanse til bruk i `getActionMap()` metoden. Som andre parameter i sistnevnte metode oppretter vi et Action objekt av en tilhørende privat klasse, som igjen styrer hvilken handling som skal utføres ved bruk av den aktuelle tastatursnarveien, se eksempel 3.20.

Eksempel 3.20: Oppsett av tastatur snarveier.

```
1      annonseVindu.getToppanelAnnonse().getSokeKnapp().getInputMap(  
        JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke("ENTER"), "enterPressed");  
2      annonseVindu.getToppanelAnnonse().getSokeKnapp().getActionMap().put("enterPressed", new EnterActionAnnonse());  
3  }  
4  
5  private class EnterActionAnnonse extends AbstractAction {  
6  
7      @Override  
8      public void actionPerformed(ActionEvent e) {  
9          annonseVindu.getToppanelAnnonse().getSokeKnapp().doClick();  
10     }  
11 }
```

3.7.12 Lyttere mellom forskjellige kontrollere i programmet

Kontrollerne er i hovedsak uavhengig av hverandre. Det vil si at `ControllerTabell.java` ikke kan kontakte `ControllerToppPanelMegler.java` direkte. Selv om denne problematikken i akkurat dette tilfelle kunne løses ved bruk av å teste på tabellens `getModel`-metode og finne ut hvilken data som ligger i tabellen, så har vi tatt vare på denne implementasjonen som ble brukt før vi hadde en solid tabell-implementasjon.

Det finnes flere eksempler som fungerer på tilsvarende måter andre steder i programmet, men det er lettest å visualisere dette eksempelet.

Målet er altså at i det øyeblikket et søk blir gjort i `ToppPanelet` så skal resultatet sendes til `ControllerTabell.java` for å settes inn i rett `TabellModell`. Vi har et interface som heter `ListListenerInterface.java` som har to metoder. Den ene metoden tar i mot en `ArrayList` og et Enum-objekt. `ControllerToppPanelAnnonse.java` har en metode som følgende (eks 3.21):

Eksempel 3.21: `setListListener`-metoden fra `ControllerToppPanelAnnonse.java`

```
1      public void setListListener(ListListenerInterface listListener) {
```

```

2      this.listListener = listListener;
3  }

```

Metoden blir kalt opp allerede i konstruktøren til `MainController.java` under oppstart av programmet, via denne instruksjonen:

Eksempel 3.22: Setter lytter fra `MainController.java`

```

1      toppPanelControllerAnnonse.setListListener(new ListListenerInterface()
2          {
3          @Override
4          public void listReady(ArrayList liste, ObjectType obj) {
5              //Brukes ikke her
6          }
7
8          @Override
9          public void listReady(HashSet liste, ObjectType objekttype) {
10             tabellControllerAnnonse.tomTabellOgKlargjorForNyttDatasett();
11             tabellControllerAnnonse.settInnDataITabell(liste, objekttype);
12             liste.clear();
13         }
14     });

```

Foreløpig er det ikke en fungerende løsning. I `sendSokeResultat`-metoden i `ControllerToppPanelAnnonse.java` så ligger følgende instruksjon helt til slutt:

Eksempel 3.23: Utdrag fra `sendSokeResultat`-metoden i `ControllerToppPanelAnnonse.java`

```

1
2      else if (listListener != null) {
3          listListener.listReady(sokeResultat, radioTypeValgt);
4      }

```

Det som skjer nå er at hver gang det blir gjort et søk så kalles lytteren `listListener` sin `listReady`-funksjon opp, der det sendes med søkeresultatet og en identifikator på hvilken type objekter som finnes i søkeresultatet. Hvis man ser på kodeeksempel 3.22 så er det her det nå kommer inn et datasett fra `Topppanelet` og det sendes da videre til rett metode i `ControllerTabell`. Her har vi altså brukt et `interface` til å være bindeledd mellom komponenter som eller ikke kjenner til hverandre. Det samme gjøres i utstrakt bruk i tabellens «museklikk»-metoder, både ved høyreklikk og dobbelklikk er det hengt på en `lytter` som vet når noe er utført og en `event` skal utføres.

3.8 Søk

Generelt i programmet finnes det to måter å søke data på: (1) **fritekstsøk** som kan foretas av megler samt (2) **annonsefilter** som foretas av boligsøker. Megleren begrenser sitt søk til et spesifikk register f.eks. boligregister eller utleierregister, deretter mottar søkeresultat basert på den tekst som er skrevet inn i tekstfeltet. Megleren kan også hente opp en full¹ registerliste

¹Foreløpig er det ikke implementert en begrensning her, dersom programmet skal brukes til store datamengder borde det innføres en begrensning på antall rader som kan hentes opp til tabellen.

gjennom å bruke et tomt søkefelt eller en stjerne «*» som søkeparameter. En boligsøker søkeresultat filtrert istenden, dette gjøres gjennom å bruke kriterier som poststed, boligtype, boligstørrelse [m^2] og utleiepris. Dersom søkeren ikke bruker boligstørrelse eller utleiepris om parameter vil annonsene blir filtrert på kun poststed og boligtype. Søkeklasser er plassert i `search` pakken og består av `FreeTextSearch.java` (megler) samt `AnnonseFilter.java` (boligsøker).

3.8.1 Meglersøk

Fritekstsøk for megler består i bunn av en klasse og et interface:

`FreeTextSearch.java` klassen bruker en iterator for søk igjennom en generisk `HashSet` over klasser som implementerer interface `Searchable`

`Searchable.java` er et interface som er påkrevd av alle klasser som skal være mulig å bruke `FreeTextSearch` på. iterfacet krever at hver klasse implemetrer metoden `toSearch()` som returnerer en `String[]` over datafelt i klassen som man skal kunne søke tillempe søket på.

Fritekstsøk blir gjennomført på en av registrene som brukeren velger, f.eks. boligregister, utleierregister, annonseregister, mm. Søket foretas over datafelt i de objekt som inngår i et spesifikt register. For at en klasse skal kunne inngå i søket må den implementere interface `Searchable` som i sin tur krever en metode som returnerer `String[]`. Arrayen skal innholde en `toString()` representasjon av de datafelt som skal inngå i søket (eksempel 3.24 og 3.25 viser oppbygging interfacet og implementering av metoden `toSearch()`). En slik implementering gjør det enkelt å legge til søkemetoder uten å bruke mye kode som henter datafelt via klassens `get` metoder. For at klassen skal inngå i søket må den nå selv returnere sine felt i streng format. Dersom man i fremtiden ønsker å inkludere flere datafelt i klassene som skal inngå i søket kan disse på en enkel måte implementeres i søket gjennom bruk av `Searchable` interfacet og `toSearch()` metoden.

Eksempel 3.24: Oversikt over `Searchable` interface

```
1 public interface Searchable {
2     String[] toSearch();
3 }
```

Eksempel 3.25: Implemntasjon av metode `toSearch()` i klassen `Person.java`

```
1 @Override
2 public String[] toSearch(){
3     String[] searchFields = {
4         String.valueOf(personID),
5         fornavn,
6         etternavn,
7         epost,
8         telefon};
9
10    return searchFields;
11 }
```

I eksempel 3.26 presenteres selve itereringen over objektene i et generisk `HashSet` som implementerer interfacet `Searchable`. Metoden `searchForPattern()` tar inn generisk `HashSet` med begrensning for interface og en tekststreng som inneholder en søkeparameter. Den innkommende søkeparameteren blir rensset på eventuelle blanksteg og satt til «lowercase». Hvis søkeparameteren ikke er tom så itererer vi gjennom mengden som ble mottatt via interface `Searchable`. For hver treff blir array med datafelt i form av `String` gjennomført etter treff med metoden `contains(String s)`.

Eksempel 3.26: Iterasjon over generisk `HashSet` som implementerer interface `Searchable`

```

1  public ArrayList<T> searchForPattern(HashSet<? extends Searchable> liste,
    String pattern) {
2
3      pattern = pattern.trim();
4      pattern = pattern.toLowerCase();
5
6      if (liste != null) {
7          if (pattern.equalsIgnoreCase("søk") || pattern.equals("") ||
            pattern.equals("*")) {
8              for (Searchable o : liste) {
9                  resultList.add((T) o);
10             }
11         } else {
12             for (Searchable o : liste) {
13                 checkMeForResults = o.toSearch();
14
15                 for (String s : checkMeForResults) {
16                     s = s.toLowerCase();
17                     if (s.contains(pattern)) {
18                         resultList.add((T) o);
19                     }
20                 }
21             }
22         }
23         return resultList;
24     } else {
25         System.out.println("En tom liste ble sendt inn til søkemetoden");
26         return null;
27     }
28 }

```

Vi er kjent med begrensninger som finnes i denne søkemetoden, disse diskuteres i detalj i avsnitt 5.2.1, side 79.

3.8.2 Annonsefilter

I denne seksjon følger en beskrivelse av `AnnonseFilter.java`. Konstruktøren til klassen tar i mot et `HashSet` over annonse objekter (`HashSet<Annonse>`). Klassen internt jobber med tre `HashSet` som brukes mellom algoritmens filtreringstrinn, disse mengder blir initialisert i konstruktøren til klassen (se eksempel 3.27). Metoden `annonseListeTmp` brukes mellom filtreringstrinn og `annonseListeFiltrert` blir brukt til å ferdigfiltrere resultat.

Eksempel 3.27: `AnnonseFilter.java`: Konstruktør

```

1  public AnnonseFilter(HashSet<Annonse> annonseliste) {
2      this.annonseListeOriginal = annonseliste;
3      annonseListeFiltrert = new HashSet<>();
4      annonseListeTmp = new HashSet<>();
5  }

```

Uansett hvis brukeren legger inn alle filtreringsparametre eller ikke, kommer filtreringen alltid til å foregå internt etter en fordefinert rekkefølge: (1) poststed, (2) boligtype, (3) utleiepris, (4) boligstørrelse samt (5) resterende parametre som fellesvask, hage og kjeller. Alle de parametrene blir sendt til filtreringmetoden fra kontroller for TopPanel i AnnonseArkFane. Selve interne filtreringsalgoritmen foretar følgende trinn ved filtrering:

1. For hver element i `annonseListeOriginal` kontroller dersom `poststed` stemmer og kopier elementet til `annonseListeTmp`. Etter at ha iterert over alle elementer gå kopier `annonseListeTmp` til `annonseListeFiltrert`.
2. For hvert element i `annonseListeFiltrert` kontrollere dersom `boligtype` stemmer og kopier hver da elementet til `annonseListeTmp`. Etter at ha iterert gjennom alle elementer i `annonseListeFiltrert` skriv over den listen med innhold fra listen `annonseListeFiltrert`.
3. Foreta samme tilnærmingsmåte som i trinn 2 for alle søkeparametre.
4. Returner filtrert set `HashSet<Annonse> annonselisteFiltrert`.

I eksempel 3.28 presenteres den rekkefølge som foretas på intern filtrering. Metoden kaller opp flere interne metoder som foretar filtrering etter hver enkel parameter sendt fra kontrolleren. Eksempel 3.29 vises hvordan filtreringen foretas i en privat metode. I dette eksemplet blir data filtrert på *min* og *maks* pris. Filtrering i de øvrig private metodene blir foretatt på en relativt tilnærmet måte (henviser spesielt interesserte til å eventuelt betrakte løsningen i selve kildekoden til oppgaven, pakke `search`, fil `AnnonseFilter.java`).

Eksempel 3.28: `AnnonseFilter.java`: Filtreringsrekkefølge etter mottatte parametre.

```

1  public HashSet<Annonse> filtrerEtterParametre(String poststed, Boligtype
    boligtype, int prisMin, int prisMaks, int arealMin, int arealMaks,
    boolean harBalkong, boolean harFellesvask, boolean harHage, boolean
    harKjeller) {
2      filtrerEtterPostSted(poststed);
3      filtrerEtterBoligType(boligtype);
4      filtrerEtterPrisRange(prisMin, prisMaks);
5      filtrerEtterBoArealRange(arealMin, arealMaks);
6      if (harBalkong) {
7          filtrerEtterBalkong();
8      }
9      if (harFellesvask) {
10         filtrerEtterFellesvaskeri();
11     }
12     if (harHage) {
13         filtrerEtterHage();
14     }
15     if (harKjeller) {
16         filtrerEtterKjeller();
17     }
18     return getFilteredResults();

```

19 }

Eksempel 3.29: AnnonseFilter.java: Eksempel på privat filtreringsklasse etter bolig areal som foretas mellom trinn for pris og balkong.

```

1      private void filtrerEtterBoArealRange(int min, int maks) {
2          if (min != 0 && maks != 0) {
3              for (Annonse a : annonseListeFiltrert) {
4                  if (a.getBolig() != null) {
5                      if (min <= a.getBolig().getBoAreal() && a.getBolig().
                        getBoAreal() <= maks) {
6                          annonseListeTmp.add(a);
7                      }
8                  }
9              }
10         kopierTilFiltrerteResultat();
11     }
12 }
```

3.9 Bilder

Bilder i programmet følger med boligobjektet, bildene blir heller ikke lagret i selve objektet og serialiseres derfor ikke. Referanse til bildemappen er lik static variabeln `objektID`. Første bilde som brukeren laster opp blir brukt på fremsiden ved presentasjon av boligen. Fremsidebilde blir plassert i boligmappen som 1.jpg. Men dersom brukeren velger å laste opp flere bilder får de inkrementelle navn som 2.jpg, 3.jpg og så videre.

3.9.1 Bildeklasser

Bildefilsti.java

Primær oppgave for klassen er å sørge for at programmer får en absolutt filsti til hvor alle bilder er lagret uavhengig operativsystem. Metodene i de klasse returnerer en **String** som benyttes i `File()` objekter i flere klasser i programmet. Klassen ble opprettet på grunn av at html visninger stiler krav på en absolutist filsti for `html` som begynner med en «file:» prefiks. Derfor metoder i klassen kan:

- Returnere plassering til `programdata/img/` uavhengig operativsystem.
- Returnere filsti til et standard bilde som vises dersom brukeren ikke laster opp egne bilder.
- Gitt et boligobjekt:
 - Filsti til boligens fremsidebilde som kan brukes i `File()` og som html sti²
 - Filsti til boligens gallerimappe og som html sti

²Som beskrevet over blir strengen returnert med prefix «file:»

BoligBilde.java

Klassen brukes til bildebahendling og opplasting av bilder til eksisterende boligobjekter. Metoder i klassen (1) kontrollerer antall allerede opplastede bilder for boligen, (2) leser inn nytt bilde fra harddisk (gitt et `File()` objekt), (3) endrer størrelsen på bildet slik at det blir tilpasset den visning som brukes i programmet (samt for å holde størrelsen nede på bildemappen), (4) lagrer behandlet bile i gallerimappen for boligen.

3.9.2 Lagring av bilder

Lagring av bilder er ikke serialisert hvilket medfører at alle bilder blir lagret som bildefiler i `programdata/img/boligbilder/objektID`. For hvert nytt boligobjekt som blir opprettet lages det en ny mappe med samme navn som `objektID` i `programdata/img/boligbilder/`. Dette gjør at serialisering av bildeobjekter er ikke nødvendig, deretter start og avslutning av programmet blir betydelig raskere³. Etter at bildemappe er opprettet kan brukeren legge til bilder for boligen. Det blir foretatt følgende trinn ved opplasting av et bilde for en bolig:

1. Fra vindu for boligregistrering/endring brukereren initierer `JFileChooser`, dersom man velger å laste opp bilder. Her blir det gjort forskjell på hvis dette er en ny, eller en eksisterende bolig. Hvis det er en ny bolig blir brukeren presentert med en dialog som spør om opplasting av bilder etter at registreringen blitt foretatt.
2. Dersom brukeren laster opp en ny fil blir den buffret som `BufferedImage` og skalert til mindre dimensjoner, eksempel 3.30 og 3.31.
3. Etter endring av størrelse blir bildet lagret med et inkrementelt filnummer (forrige bilde som ble lastet opp + 1). Dersom det ikke finnes tidligere bilder blir det første opplastede bilde satt til 1.jpg, eksempel 3.32.

Eksempel 3.30: BoligBilde.java: Innlesning av bildefil

```
1 private void lesInnBilde(File bildeFil) throws IOException {
2     bilde = ImageIO.read(bildeFil);
3     bildeType = bilde.getType() == 0 ? BufferedImage.TYPE_INT_ARGB : bilde
        .getType();
4 }
```

Eksempel 3.31: BoligBilde.java: Endring av opplastet bildestørrelse

```
1 private BufferedImage endreBildeTilStandardStorrelse(BufferedImage
    originalBilde, int bildeType) {
2     BufferedImage nyttBilde = new BufferedImage(Konstanter.BILDE_WIDTH,
        Konstanter.BILDE_HEIGHT, bildeType);
3     Graphics2D grafikk = nyttBilde.createGraphics();
4     grafikk.drawImage(originalBilde, 0, 0, Konstanter.BILDE_WIDTH,
        Konstanter.BILDE_HEIGHT, null);
5     grafikk.dispose();
6     return nyttBilde;
7 }
```

Eksempel 3.32: BoligBilde.java: Lagring av et nytt eller tilleggsbilde for en bolig

³Avhengig av hvor mange bolig objekter som er lagret i registrene.

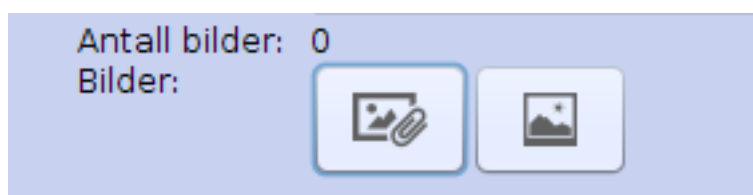
```
1 private void lagreBilde(BufferedImage bilde, String path) throws IOException
2     {
3         ImageIO.write(bilde, "jpg", new File(path));
4     }
5 public void lagreNyttBildeForBolig(Bolig bolig, File innlestFil) throws
6     IOException {
7     lesInnBilde(innlestFil);
8     BufferedImage tmpBilde = endreBildeTilStandardStorrelse(bilde,
9         bildeType);
10    String galleriSti = getGalleriSti(bolig);
11    String fullSti = galleriSti + "/" + String.valueOf(getNesteFilnummer(
12        bolig)) + ".jpg";
13    lagreBilde(tmpBilde, fullSti);
14 }
```

3.9.3 Visning av bilder

Bildevisning kan initialiseres på to måter:

1. Fra vindu for boligbehandling, figur 3.2 der megleren får mulighet til å laste opp nye boliger eller starte en bildevisning for allerede registrerte boliger. Dersom brukeren laster opp et bilde blir man også presentert med en dialog med spørsmål om å registrere flere bilder på samme bolig, se figur 3.3. Det kan registreres frivillig antall bilder for en bolig.
2. Den andre og siste metoden for å starte en bildevisning er å klikke i visningsvindu til boligsøker. Dette vil kalle opp visningsvindu for bildene, se 3.4.

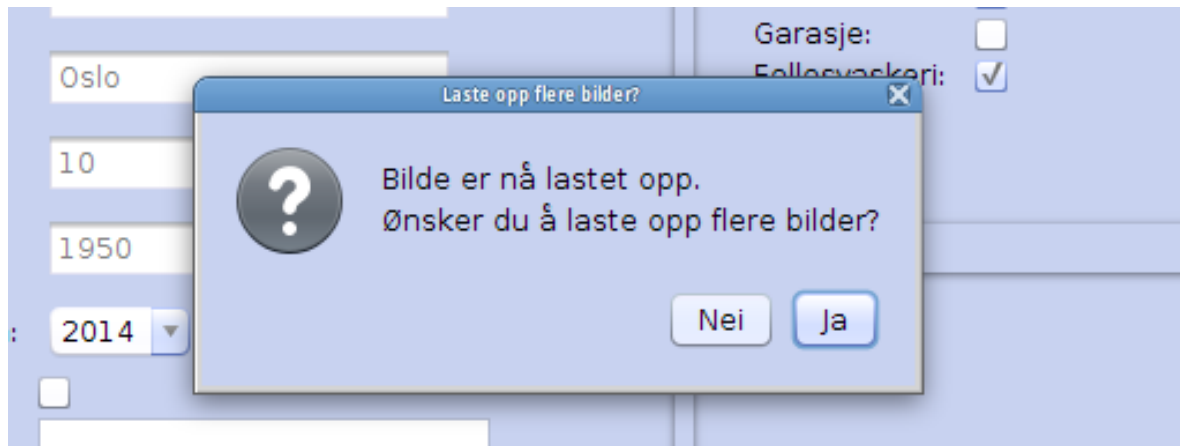
Dersom brukeren velger å ikke laste opp minst et bilde for en bolig blir kommer metoden `BoligBilde.antallBilder` til å rapportere 0 og derigjennom blir det hentet et standardbilde fra `./programdata/img/default/1.jpg` og presentert både i annonser og i bildevisningen. Se figur 3.5.



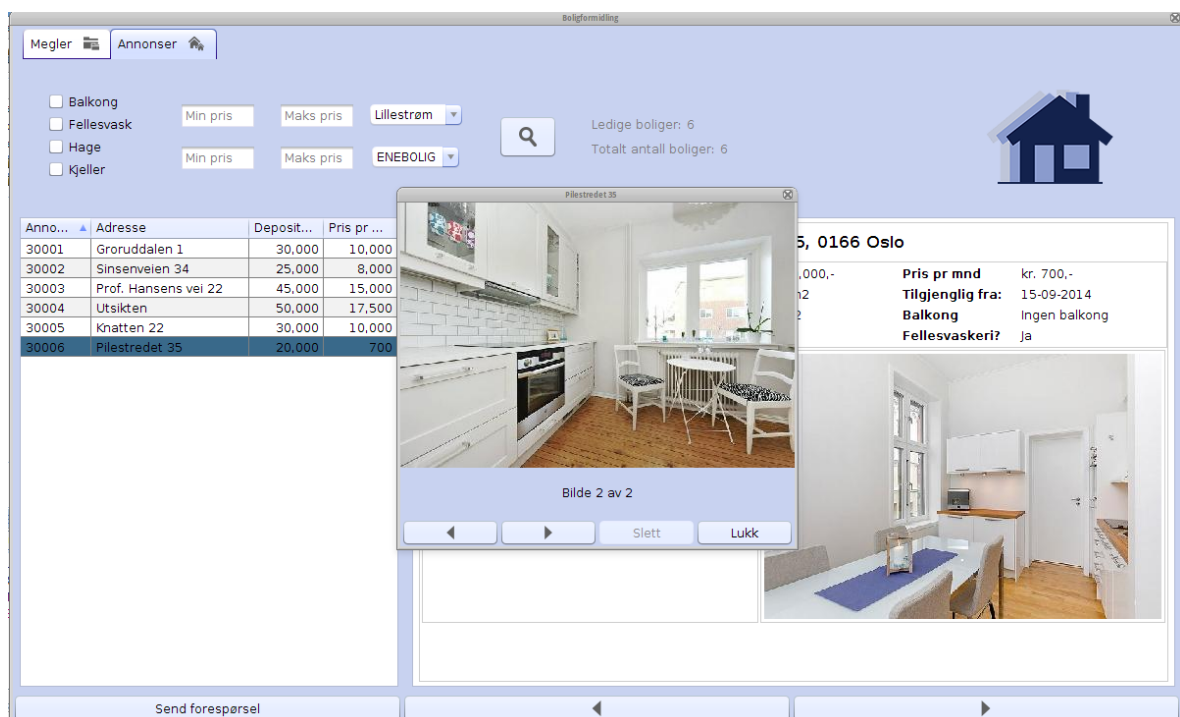
Figur 3.2: Utsnitt fra boligbehandlingsvindu. Viser kontroller for opplastning og visning av bilder som er registrert for dette bildeobjektet.

3.9.4 Sletting av bilder

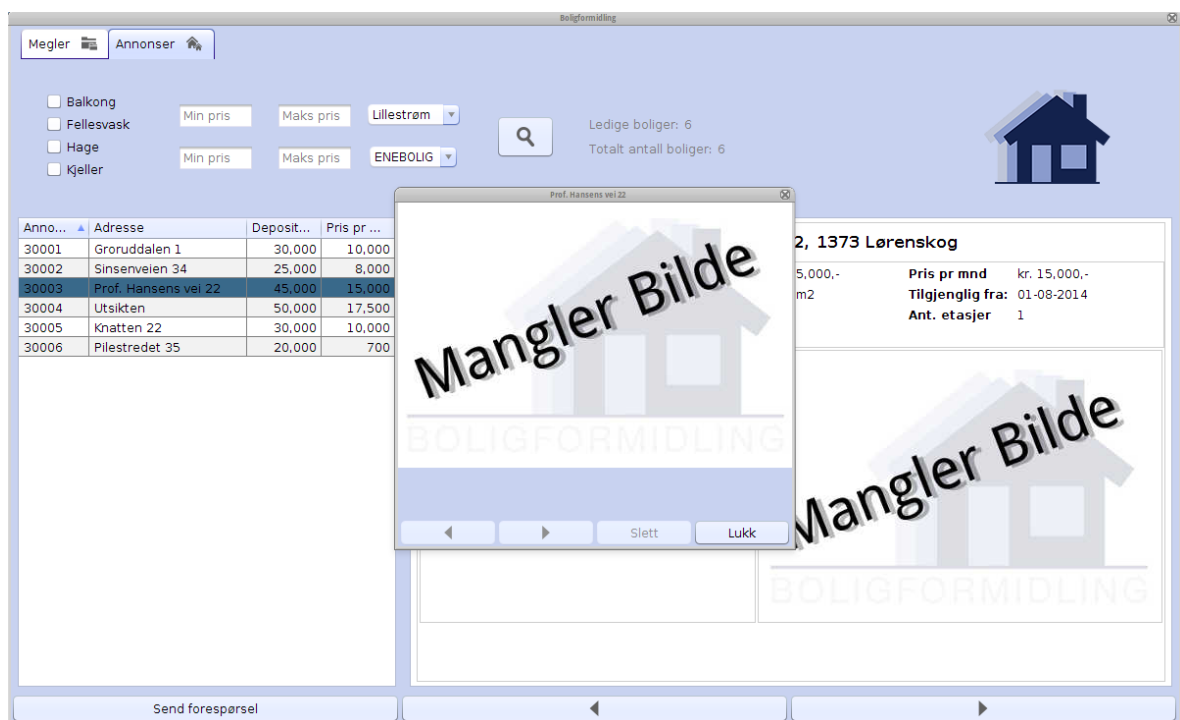
Sletting av bilder er foreløpig ikke implementert. Dersom et boligobjekt blir slettet må gallerimappen til boligen slettes manuelt. Det er tatt høyde for å implementere sletting og mulighet for dette er satt opp i brukergrensesnittet men foreløpig er deaktivert.



Figur 3.3: Utsnitt fra boligbehandlingsvindu. Viser forespørsel til bruker (megler) dersom den ønsker å laste opp flere bilder for boligobjetet.



Figur 3.4: Bildevisning initialisert gjennom klikk i visningsarea for boligsøker.



Figur 3.5: Eksepel på standardbilde som vises dersom brukeren ikke har lastet opp et bilde etter registrering av en ny bolig.

3.10 Konstanter og Enum

Det er brukt konstanter og flere enum typer i programmet, disse finner man i `lib` pakken. Konstanter er brukt i form av publike statiske variabler som er tilgjengelige tvers over alle pakker og klasser. Eksempel på dette er regex konstanter og konstanter som f.eks. brukes til å sette opp størrelse på GUI komponenter som vinduer eller tekstfelt. Enum for å identifisere instanser av klasser eller klassevariabel isteden for å teste f.eks. på en streng som en identifikator. Eksempel på dette er data i en `comboboks`. Dersom vi fyller komboboksen med enumtyper kan vi direkte teste på hva som returneres til kontrollern fra gui isteden for å ta opp å bruke `equals()` metoden. Enum gir oss også mulighet til implementering av metoder direkte i enum klassen som kan foreta eventuelle beregninger på sine klassevariabler.

3.10.1 RegexTester.java

Klassen er satt sammen med hensikt å definere alle regex tester som foretas i programmet (oftest ved inhentning av data lagt inn fra bruker). Klassen består av publike regex konstanter som kan etter behov hentes opp av metoder der man kun etterspør en streng med et regex mønster, f.eks `CustomJTextField`. Den andre delen av klassen består av boolean metoder som «speiler» alle regex streng konstanter og foretar et test på valgt regex konstant og en mottatt tekststreng som parameter. I eksempel 3.33 presenteres en regex streng som brukes til å kontrollere en gateadress. Adressen må begynne med en stor bokstav, kan være opp til tre ord i lengden, deretter må avsluttes med et nummer opp til tre tall som kan blir etterfulgt av blankt steg og en bokstav (f.eks. trappeoppgang).

Eksempel 3.33: Regexstreng for gateadresse og husnummer.

```
1 public static final String GATE_ADRESSE = "[A-ZÆØÅ]{1}[a-zæøå]{1,20}[\\s]?[A-ZÆØÅ]?[a-zæøå]*[\\s]?[A-ZÆØÅ]?[a-zæøå]*[\\s][1-9]{1}[0-9]{0,2}?[\\s]?[A-ZÆØÅ]{0,1}$";
```

Eksempel 3.34 presenteres privat metoden som bruker Java sin interne regex metode for å foreta test på mønsteret. Metoden blir kalt opp fra intern klasse som er spesiallaget for hver av regex konstantene (se neste eksempel).

Eksempel 3.34: Private regex test metode.

```
1 private static boolean patternMatchOK(String input, String regex) {
2     try {
3         erTestOK = input.matches(regex);
4     } catch (PatternSyntaxException e) {
5         System.out.println("Regex xception: input = " + input + " regex = " + regex);
6     }
7     return erTestOK;
8 }
```

I eksempel 3.35 presenteres som «speiler» hver enkelregex string konstant i klassen. Følgende metoder brukes i kontrollene som et andre kontroll trinn da et nytt objekt skal opprettes og legges til i registeret (første testet blir normalt foretatt i `CustomJLabel` med visuell *feedback*). Ettersom metoden er `static` trenger den ikke å bli initialisert.

Eksempel 3.35: Static regex metode til tilhørende regex møsnter streng.

```

1  public static boolean testGateadresse(String gateAdresse) {
2      return patternMatchOK(gateAdresse, GATE_ADRESSE);
3  }

```

3.10.2 Konstanter.java

Klassen består av flere publike konstanter lagret som static variabler som er tilgjengelige for alle klasser i programmet. I eksempel 3.36 presenteres noen av de konstantene som er satt opp i klassen som viser et kort utdrag for å presentere strukturen som brukes gjennom hele klassen.

Eksempel 3.36: Noen av static konstanter som brukes i Konstanter klassen.

```

1  /**
2   * Kollator rekkefølge som brukes til sortering.
3   */
4  public static final String KOLLATOR_REKKEFOLGE = "<\0<0<1<2<3<4<5<6<7<8<9"
5      + "<A,a<B,b<C,c<D,d<E,e<F,f<G,g<H,h<I,i<J,j"
6      + "<K,k<L,l<M,m<N,n<O,o<P,p<Q,q<R,r<S,s<T,t"
7      + "<U,u<V,v<W,w<X,x<Y,y<Z,z<E,æ<Ø,ø<Å=AA,å=aa;AA,aa";
8
9  /**
10   * Felles serialiseringsnummer som brukes til unik nummer ved lagring av
11   * programmets datastruktur.
12   */
13  public static final long SERNUM = 1234L;
14
15  /**
16   * En relativ path til alle eksterne filer som brukes i programmet som
17   * serialisert data, bilder osv.
18   */
19  public static final String PROGRAMDATA = "programdata";
20
21  /**
22   * Serialisert fil som brukes til lagring og innlesning av all data i
23   * programmet.
24   */
25  public static final String FILNANV = Konstanter.PROGRAMDATA + "/data.iso";

```

3.10.3 GUI konstanter

Følgende er konstanter som brukes for å sette opp en fordefinert størrelse på alle komponenter i brukergrensesnittet. De består av to klasser:

VinduStorrelse.java En enum klasse som returnerer alle størrelse på vinduer som brukes i programmet. Den kan også brukes for å hente opp kun bredde eller høyde for et spesifikk vindu. Eksempel over klassens datafelt og konstruktør presenteres i eksempel 3.37.

GuiSizes.java En klasse med konstanter som brukes til nå sette opp interne swing komponenter som f.eks. bredde på CustomJTextField eller CustomJButton.

Ikoner.java Brukes til å hente opp referanser til alle ikoner som brukes tvers i hele programmet. Static konstanten blir satt opp som en `ImageIcon` variable som deretter kan brukes direkte for å sette opp et bilde i en `JPanel` eller liknende. Et kort eksempel over hvordan klassen er satt opp finnes i eksempel 3.38.

Eksempel 3.37: Enum klasse for vindustørrelser

```

1 public enum VinduStorrelse {
2
3     STOR (730, 1200),
4     MIDDEL (600, 800),
5     LITEN (300,400),
6     TOPPANEL (150,0),
7     BUNNPANEL (30,0),
8     VENSTREPANEL (0,400),
9     SENTERPANEL (0,0);
10
11     private final int WIDTH;
12     private final int HEIGHT;
13
14     private VinduStorrelse(int HEIGHT, int WIDTH) {
15         this.WIDTH = WIDTH;
16         this.HEIGHT = HEIGHT;
17     }
18     ...
19 }

```

Eksempel 3.38: Utsnitt fra konstantklasse med static variabler for programikoner.

```

1 public class Ikoner {
2
3     private final static String ikonerSti = new BildeFilSti().
4         getAbsoluteGalleryPath() + "/default/ico/";
5     ...
6     //Tabs, 24px, 4px padding, farve 606060
7     public final static ImageIcon ANNONSER = new ImageIcon(ikonerSti + "Houses
8         -24.png");
9     public final static ImageIcon MEGLER = new ImageIcon(ikonerSti + "Folder-
10         Copy-24.png");
11     ...
12     //Applikasjonsikoner, 128px, 0px padding, E8E8E8
13     public final static ImageIcon APP_ICON = new ImageIcon(ikonerSti + "
14         boligLogo.png");
15     public final static ImageIcon NY_UTLEIER = new ImageIcon(ikonerSti + "
16         ny_utleier.png");
17     public final static ImageIcon NY_BOLIG = new ImageIcon(ikonerSti + "
18         ny_bolig.png");
19     ...
20 }

```

3.10.4 Enum

I `lib` pakken er det satt opp flere klasser av enum type, en enum klasse ble forklart i avsnitt 3.10.3. De resterende enum klassene som brukes i programmet er:

Boligtype.java Definerer de forskjellige boligtypene som: *Enebolig*, *Tomannsbolig*, *Rekkehus*, *Leilighet*, *Andre*. I programmet i dagens dato brukes det kun *Enebolig* og *Leilighet*. Enum klassen tar dog forbehold for videreutvikling av programmet gjennom å inkludere andre boligtyper.

Sivilstatus.java Brukes for populering av kombobokser ved registrering av en ny boligsøker. Enum typen blir distribuert direkte mellom komboboks og kontroller.

Arbeidsforhold.java Fungerer på samme måte som **Sivilstatus.java** og brukes til sammen funksjoner i programmet med tar for seg boligsøkerens arbeidsforhold.

Objekttype.java Brukes til å definere hvilken objekttype som sendes over i flere transaksjoner i programmet. Et eksempel på dette er renderering av tabell som blir satt etter hvilket objekttype som er definert i enum. Klassen spesifiserer objekttype på en øvre nivå, hvilket betyr at objektene blir spesifisert på superklasse nivå. Eksempelvis gjør denne enum typen ingen forskjell på objekttype *Utleier* eller *Leietaker* uten kan kun vise at objektet som sendes over er av type *Person*.

Objekttype2.java Fungerer og brukes på samme måte som **Objekttype.java** med inneholder detaljert informasjon over hvilke objekter som kan passerer mellom transaksjonene. Her gjør vi altså forskjell mellom underliggende klasser som *Utleier* og *Leietaker*.

3.11 Tilpassede Swing komponenter

I programmet er det brukt flere spesialtilpassede komponenter arvet fra swing klassen der vi har spesifisert størrelse, brukerinteraksjon eller andre tilpasninger for å slippe å gjøre dette hver gang en slik komponent brukes. Komponentene er plassert i pakke **view** og **view.register**. I dette avsnittet beskrives spesielt tilpassede komponenter som har spesifikk betydning for programmet.

3.11.1 AbstractPanel.java

Abstractpanel som arver **JPanel** er klassen som ligger til grunn til alle paneler som er i bruk i programmet. Klassen har to konstruktører og har som regel følgende oppgaver:

- Setter størrelse (dimensjon) på panelet.
- Setter bakgrunnfarge på panelet fra en global static konstant. Dette gjør at farge på all brukergrensesnitt i programmet enkelt kan endres.
- Sette en **titleLabel** rundt komponenten med tittel fra den innkomne parameter **tittel**.

Dimensjonen settes ved å kalle opp metoden **setPreferredSize(Dim dim)** som brukes ettersom slik tilnærming gjør at størrelse på panelet blir «respektert» av valgt layout manager (dette gjelder ikke alltid dersom **setSize()** brukes).

3.11.2 MainPanel.java

MainPanel er klassen som arver AbstractPanel og setter opp både megler og annonse-panelet, se eksempel 3.39. Arv fra superklassen består av at man kaller opp en tom konstruktør i superklassen som setter opp bakgrunnfarge. Deretter blir det satt opp en enkel GridLayout som består av én celle. Det vil si den dekker hele vinduet. Det legges til en JTabbedPane som legger til panel for annonse og megler. Annonsepanelet og meglerpanelet er igjen to klasser det kommer tilbake til i de neste avsnittene. Klassen MainPanel blir initialisert fra StartGUI.java ved oppstart av programmet, der de ferdigopprettede ArkfaneMegler.java og ArkfaneAnnonse.java sendes med som parametere for å kunne legges inn i JTabbedPane.

Eksempel 3.39: Konstruktør i MainPanel.java

```

1  public MainPanel(AbstraktArkfane megler, AbstraktArkfane annonse){
2      setLayout( new GridLayout( 1, 1) );
3      this.megler = (JPanel) megler;
4      this.annonse = (JPanel) annonse;
5
6      arkfaner = new JTabbedPane(JTabbedPane.TOP);
7
8      //Legger til tab og kobler med panelet.
9
10     arkfaner.addTab("Megler  ", Ikoner.MEGLER, this.megler);
11     arkfaner.addTab("Annonser  ", Ikoner.ANNONSER, this.annonse);
12
13     arkfaner.setSelectedIndex(1);
14     arkfaner.setToolTipTextAt(0, "Administrering av boliger, søknader mm."
15                               );
16     arkfaner.setToolTipTextAt(1, "Finn tilgjengelige boliger, send inn sø
17                               knader mm.");
18     add(arkfaner);
19 }
```

3.11.3 AbstraktArkfane.java

AbstraktArkfane.java arver AbstractPanel.java. Klassene som arver AbstracrArkfane.java får et oppsett av paneler. Hvilket oppsett av paneler som blir opprettet er avhengig av hvilken parameter som blir sendt inn i konstruktøren, se eksempel 3.40. Hvilken type av arkfane som skal bli opprettet bestemmes a strengparameteren som blir sendt inn til konstruktøren. Klassen arves av to klasser: ArkfaneMegler.java og ArkfaneAnnonse.java, som utgjør de to arkfanene i programmet.

Eksempel 3.40: Konstruktør til AbstraktArkfane.

```

1  public AbstraktArkfane(String valgtToppanel) {
2      setLayout(new BorderLayout());
3      setVisible(true);
4
5      bunnpanel = new BunnPanel(VinduStorrelse.BUNNPANEL.getHEIGHT(),
6                               VinduStorrelse.BUNNPANEL.getWIDTH());
7      venstrepanel = new VenstrePanel("Liste",VinduStorrelse.VENSTREPANEL.
8                                     getHEIGHT(),
```

```

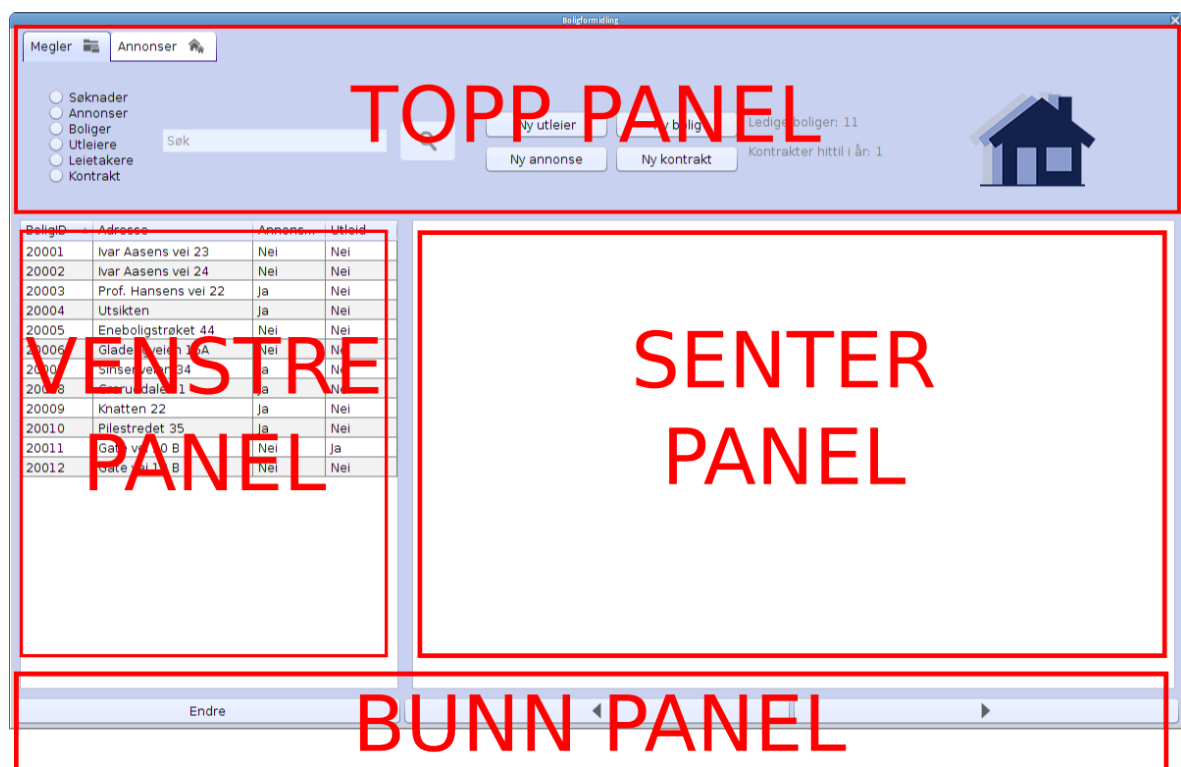
8         VinduStorrelse.VENSTREPANEL.getWidth());
9         senterpanel = new SenterPanel("Visning",VinduStorrelse.SENTERPANEL.
            getHeight(),
10            VinduStorrelse.SENTERPANEL.getWidth());
11
12         if (valgtToppanel.equals("megler")) {
13             toppanel = new TopPanelMegler("Søk",VinduStorrelse.TOPPANEL.
                getHeight(),
14                VinduStorrelse.TOPPANEL.getWidth());
15             add(toppanel, BorderLayout.NORTH);
16         } else{
17             toppanel = new TopPanelAnnonse("Søk",VinduStorrelse.TOPPANEL.
                getHeight(),
18                VinduStorrelse.TOPPANEL.getWidth());
19             add(toppanel, BorderLayout.NORTH);
20         }
21         add(venstrepanel, BorderLayout.WEST);
22         add(senterpanel, BorderLayout.CENTER);
23         add(bunnpanel, BorderLayout.SOUTH);
24     }

```

Som vi kan se i koden, AbstraktArkfane setter opp komponenter som inngår i de to visningsfanene fordelt på Annonse eller Megler. Klassen inneholder også en `LayoutManager` som setter opp alle disse komponentene med retninger: NORTH, WEST, CENTER og SOUTH. Resultatet av dette presenteres i figur 3.6 der de forskjellige panelene i klassen er satt opp. Klassen består også av et antall get-metoder som kan brukes til å returnere hele paneler til kontrollere i MVC-strukturen. Klassen blir arvet av følgende subclasser: (1) `ArkafaneAnnonse.java` og (2) `ArkfaneMegler.java`. Hver enkelt av disse subclassene kaller opp konstruktøren i superklassen `AbstraktArkfane` som initialiserer layouten og setter opp riktige paneler.

3.11.4 TopPanelMegler.java

Toppanelet i meglervinduet (figur 3.7) er kontrollert fra `ControllerToppPanelMegler.java`, komponentene i det topppanelet blir aktivert og deaktivert avhengig av hvilket register eller funksjon brukeren har valgt. Eksempel på dette er når brukeren første gang etter programstart går inn til panelet, da vil søkefeltet være deaktivert frem til riktig radioknapp velges for det register som man ønsker å søke i. Liknende funksjonalitet er lagt inn dersom man f.eks. ønsker å registrere en ny bolig. En bolig kan kun registreres på en «person» som i dette tilfelle kan være en eier eller en representant. For at man ikke skal ha mulighet til å registrere en bolig uten en eier er det allerede sperret på GUI slik at brukeren må markere en allerede registrert eier i utleierlisten og deretter klikke på «Ny bolig»-knappen. Så lenge ingen person er markert i tabellen vil knappen forbli deaktivert. Tilsvarende den funksjonaliteten er det lagt til liknende begrensninger for opprettelse av en ny annonse, da det må markeres et boligobjekt i tabellen for å få lov å opprette en ny annonse. For å opprette en nytt kontrakt må det ha ankommet en forespørsel/søknad til megleren, som først markerer forespørselen for hvilken kontrakt skal opprettes.



Figur 3.6: Fordeling mellom komponenter i AbstraktArkfane

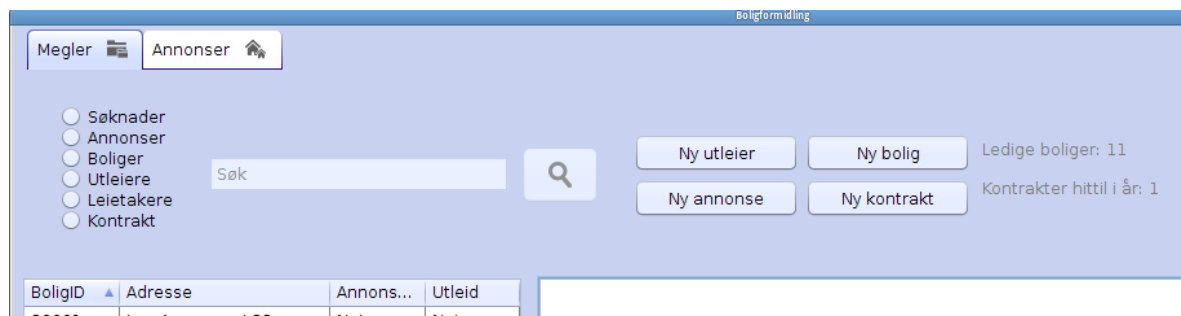
3.11.5 CustomSubPanel.java

Klassen arver den abstrakte klassen `AbstractPanel.java` (avsnitt 3.11.1). Den brukes til å sette opp indre paneler i alle registreringsvinduer som f.eks. registrering av nye boliger, utleier osv. Klassen er utstyrt med flere konstruktører som kan ta flere kombinasjoner av tittel, dimensjoner og layout manager. Slik valgfrihet gjør at panelet kan initialiseres på mange forskjellige måter og enkelt kan benyttes ut fra forskjellige behov som kan en har til brukergrensesnittet. Panelet inneholder også en metode for å ta imot en lytter (`ActionListener`) for en enkel og rask implementering sammen med en kontrollerklasse etter MVC-arkitektur.

3.11.6 CustomJTextField.java

`CustomJTextField` arver `AbstractPanel` og dermed inneholder samme konstruktør som det panelet, hvilken setter feltets dimensjoner. Slik løsning medfører også at hvert tekstfelt er plassert i en egen `JPanel`⁴. Tekstfeltet er spesialtilpasset slik at en den inneholder en indre label som kan for eksempel brukes til å sette en mal på hva brukeren skal skrive inn i tekstfeltet. Eksempelvis kan dette være forventet antall siffer i et telefon- eller personnummer, figur 3.8a. Feltet inneholder også to lyttere for `focusEvent` som initialiserer en regex etter et regex-mønster som settes via feltets konstruktør. Regex-mønster hentes via static konstanter som passer det uttrykk som feltet skal brukes til (se avsnitt 3.10.1, side 52). Etter at markøren

⁴arves av `AbstractPanel`

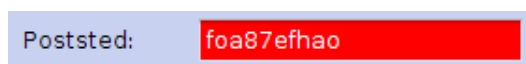


Figur 3.7: GUI komponenter i meglerpanel

flyttes ut fra feltet, og regex-testen feiler, blir feltet markert med rød farve som det vises i figur 3.8b. Panelet overrider også de metoder som man ønsker at skal være tilgjengelige fra superklassen `TextField` som er `getText()`, `setText()` og `setEnabled()`.



(a) Inaktiv, indre label



(b) Regex-kontroll feilet

Figur 3.8: Forsjellige tilstand av CustomJPanel

3.11.7 CustomJButton.java

Klassen arver `JButton` og består av totalt fem forskjellige konstruktører (se eksempel 3.41 som brukes til å sette opp en spesifikk knapp. Konstruktørene er spesifisert på en måte slik at det kan settes opp knapper med forskjellige størrelser, titler, ikoner eller også kombinasjoner av alle disse mulighetene.

Eksempel 3.41: De forskjellige konstruktørene i `CustomJButton`.

```

1 public class CustomJButton extends JButton {
2
3     private String navn;
4     private Icon ikone;
5
6     public CustomJButton(String navn) {
7         this.navn = navn;
8         setText(this.navn);
9     }
10
11     public CustomJButton(String navn, int bredde, int hoyde) {
12         this.navn = navn;
13         setText(this.navn);
14         setPreferredSize(new Dimension(bredde, hoyde));

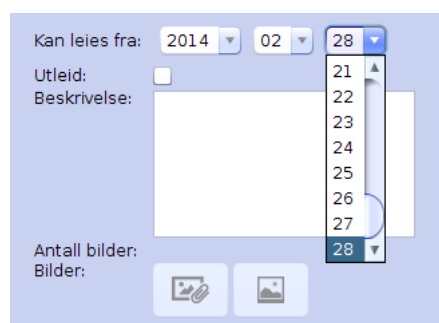
```



```
15     }
16
17     public CustomJButton(String navn, Icon ikone) {
18         this.navn = navn;
19         this.ikone = ikone;
20         setText(this.navn);
21         setIcon(this.ikone);
22     }
23
24     public CustomJButton(Icon ikone) {
25         this.ikone = ikone;
26         setIcon(this.ikone);
27     }
28
29     public CustomJButton(Icon ikone, int bredde, int hoyde) {
30         this.ikone = ikone;
31         setIcon(this.ikone);
32         setPreferredSize(new Dimension(bredde, hoyde));
33     }
34 }
```

3.11.8 ComboDatoVelger.java

Datovelger implementerer `CustomSubPanel` men hjelp av en egen layout manager. Klassen brukes med for å implementere en datovelger som skal gjøre det mulig for å velge riktig dato uten å bruke regex hvilket i sin tur skal oppleves enklere for brukeren. Komponentene består av tre `JComboBox` som brukes for valg av år, måned og deretter dag. Velgeren er utstyrt med en egen lytter som setter riktig antall dager i den siste listen etter at brukeren har valgt år og måned (se figur 3.9). Klassen kan returnere valgt data som `int` fordelt per år, måned og dag eller et `Calendar`-objekt dersom så ønskes. Dato kan også blir satt gjennom å kalle opp metoden `setDato(int ar, int mnd, int dag)` som er en funksjon som brukes ved f.eks editering av de registrerte boligene.

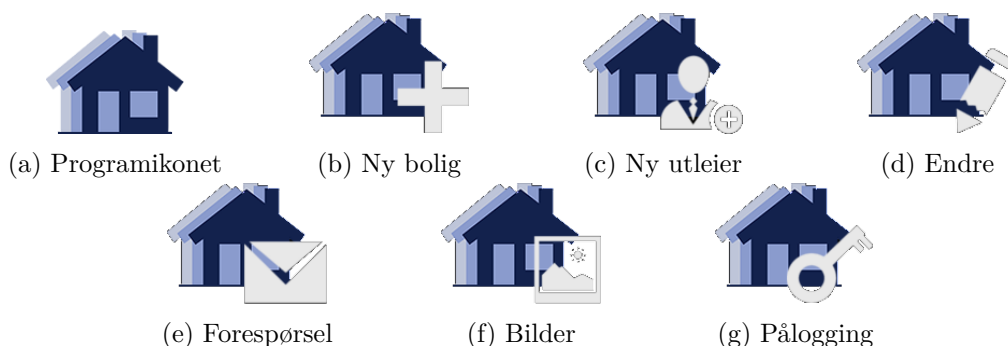


Figur 3.9: `ComboDatoVelger.java` tilpasning av antall dager.

3.12 Visuelle detaljer

3.12.1 Ikoner

Figur 3.10 presenterer alle applikasjonsikonene som brukes til vinduene i programmet. Alle ikoner til subvinduer er satt sammen av våre egne ikoner og «open source»-ikoner (se 1.5.1, side 8). Hensikten med slike komposittikoner er at de skal gi en mer visuell tilbakemelding til brukeren som viser hvilken aktuelle funksjon som brukes i programmet.



Figur 3.10: Applikasjons og vinduikoner

3.12.2 Presentasjon


Figur 3.11 viser eksempel på hvordan objektene vises i `JEditorPane` gjennom `html`-visning. Datafelt for objektet `parases` i klassen `ControllerOutput.java` i en metode som er tilpasset til visning av spesifikt objekt, f.eks boligobjektene vises gjennom `visBoligObjektHTMLOutput(Object valgtObjekt, JEditorPane output, AbstraktArkfane vindu, HashSet<Bolig> boligliste)`. Det er noen begrensninger i forhold til `html` visningen da `JEditorPane` er kun kapabel til visning av `Html` versjon 3.2 og `CSS` 1.0 fra 1997 (dette diskuteres utførlig i avsnitt 3.7.7, side 29).

3.12.3 Tabell

I figur 3.12 presenteres utseende på formatert tabellobjekt, der annenhver rad har en annen bakgrunnsfarge og får en tredje bakgrunnsfarge ved markering. Forskjellige farger brukes i tillegg til horisontale linjer med hensikt å oppnå en bedre avskilt linje mellom presentasjonen av tabellkomponenter. Alle tabeller i programmet har også inkludert en meny som hører til de forskjellige objektene utfra hvilke funksjoner i programmet som kan brukes på et vist objekt. For eksempel i en tabell over boligobjekter. Boligobjekter kan man endres, slettes eller endre publiseringsstatus. Dersom brukeren «skriver ut» innhold i utleierregisteret vil det bli presentert alternativer som er spesifikke for objekter som instansierer superklassen `person` (ny endre, slett) men i tillegg funksjoner som er spesifikke for klassen `utleier` (som presentert i den refererte figuren). I eksemplet har vi mulighet å markere en `person` direkte og gå til registreringsdialog for bolig som da registreres på den personen. Vi kan også editere boliger som tilhører denne eieren eller også slette dem. Sammen funksjonalitet er også da tilgjengelig

Leilighet					
Eier ID	10004	Bolig ID	20010		
Adresse	Pilestredet 35	Postnr	0166	Poststed	Oslo
Byggeår	1950	Boareal	300 m2	Utleid?	Nei
Etasje	2	Bodareal	10 m2	Balkong	Ingen balkong
Har heis?	Ja	Har garasje?	Nei	Fellesvaskeri?	Ja

Beskrivelse av bolig
 HIOA



Figur 3.11: HTML presentasjon av et boligobjekt.

via den vanlige og synlige gui-komponenter som knapper eller også gjennom å dobbelklikker i tabellen.

ID	Fornavn	Etternavn	Epost
10002	Hans	Pedersen	pedersen@boflott.no
10003	Petter	Stordalen	pstordalen@yahoo.com
10004	Kristian	Stormare	kstor@stormare.no
10005	Knut	Bjorøy	knut.bjoroy@bo.com
10006	Richard	Heia	richard.heia@bo.com
10013	Qwe	Qwe	qwe@bo.com
10014	Asd	Asd	ola.normann@bo.com
10016	Sdf	Sdf	ola.normann@bo.com
10018	Qwe	Qwe	ola.normann@bo.com
10021	Asdlkk	Aasdlkj	ola.normann@epost.com
10024	Zxc	Zxc	ola.normann@epost.com
10026	Ert	Ert	ola.normann@epost.com

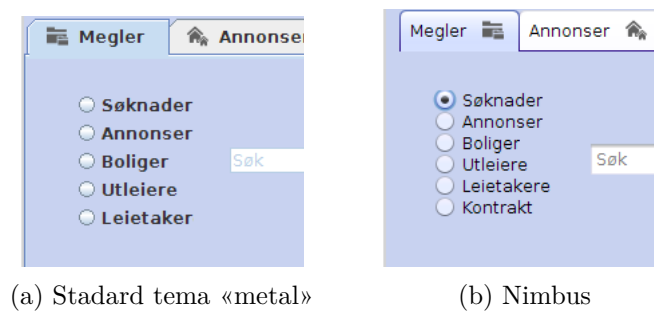
ID 10005
Navn Knut Bjorøy
Tlfnr 56320069

BoligID: 20007 **BoligID:** 20006

Figur 3.12: Visning og markering i tabell

3.12.4 Grafisk tema

Med tanke på å få bedre portabilitet mellom forskjellige operativsystem er standard Java «LookAndFeel» endret fra **Metal** til det nyeste swing tema **Nimbus**. Den primære årsaken til dette er at det ble observert noen forskjeller på størrelse og layout av gui-komponenter mellom operativsystemene som programmet var testet på. Eksempelvis, dersom programmet testes i Linux miljø finnes det ingen «native» grafisk miljø i form av komponenter som knapper, men på Mac OS og MS Windows der Java er kapabel å renderere standard knapper for systemet ble det noen forskjeller mellom slike komponenter. Bruk av nimbus som primær tema for gui komponenter gir sikkerhet at alle komponentene kommer til å bli renderert gjennom JVM hvilket gir en god portabilitet mellom operativsystemene (også beskrevet i avsnitt 2.5.7 «protabilitet», side 15).



Figur 3.13: Applikasjons og vinduikoner

Kapittel 4

Brukerveiledning

4.1 Forord

Programmet er delt opp i 2 hoveddeler - en del for boligsøkende, og en del for megler/administrasjon. Disse to delene av programmet er adskilt ved hjelp av arkfaner oppe i venstre hjørne av programmet (figur 4.1). Vinduet for boligsøkende er i hovedsak delt opp i 4 deler.

Annon...	Adresse	Depositum	Pris pr mnd
30001	Groruddalen 1	30,000	10,000
30002	Sinsenveien 34	25,000	8,000
30003	Prof. Hansens vei 22	45,000	15,000
30004	Utsikten	50,000	17,500
30005	Knatten 22	30,000	10,000
30006	Pilestredet 35	20,000	700

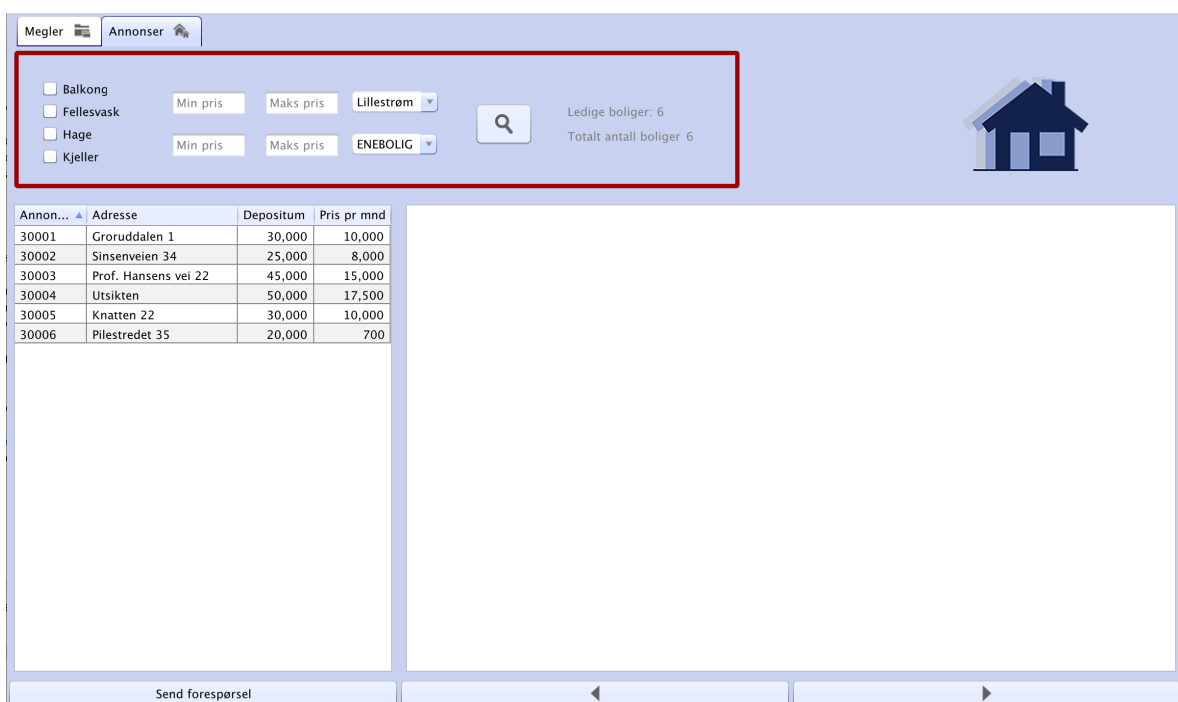
Figur 4.1: Boligsøkende.

4.2 Bolisøker

4.2.1 Filterpanel

Dette er øverste del av programvinduet markert i rødt på figur 4.2. Her har en muligheten til å legge inn ønskede søkekriterier, og søke blant tilgjengelige boliger ut ifra valgte kriterier. En kan til enhver tid trykke på “Enter” knappen for å gjøre et søk ut ifra valgte kriterier, eller ved å enkelt og greit trykke på søk-knappen.

Merk at tilgjengelige stedsnavn i dropdown menyen, vil kun vise stedsnavn for steder det er registrert boliger for. Finner du ikke stedsnavnet ditt i denne listen, så er det heller ingen tilgjengelige boliger i det aktuelle området.



Annon...	Adresse	Depositum	Pris pr mnd
30001	Goruddalen 1	30,000	10,000
30002	Sinsenveien 34	25,000	8,000
30003	Prof. Hansens vei 22	45,000	15,000
30004	Utsikten	50,000	17,500
30005	Knatten 22	30,000	10,000
30006	Pilestredet 35	20,000	700

Figur 4.2: Filterpanel.

4.2.2 Resultattabell

Dette er venstre del av programvinduet, markert i rødt på figur 4.3. Her vil resultater av et evt. utført søk vises.

En har mulighet til å sortere på annonse ID, Adresse, Depositum, og pris per mnd øverst i resultatvinduet. Du kan bla igjennom annonser ved å bruke pil opp, eller pil ned knappene, eller alternativt bruke “fram og tilbake” knappene nederst til høyre i programvinduet.

Annon...	Adresse	Depositum	Pris pr mnd
30001	Groruddalen 1	30,000	10,000
30002	Sinsenveien 34	25,000	8,000
30003	Prof. Hansens vei 22	45,000	15,000
30004	Utsikten	50,000	17,500
30005	Knatten 22	30,000	10,000
30006	Pilestredet 35	20,000	700

Figur 4.3: Resultattabell.

Trykk på et treff i tabellen for å få fram flere detaljer om den valgte boligen i...

4.2.3 Visningspanel

Her vil detaljer rundt valgt bolig (i resultattabellen) vises, markert i rødt på figur 4.4.

Visningspanel

Filter: Balkong, Fellesvask, Hage, Kjeller. Min pris, Maks pris. Lillestrøm, ENEBOLIG. Ledige boliger: 6. Totalt antall boliger: 6.

Annon...	Adresse	Depositum	Pris pr mnd
30001	Goruddalen 1	30,000	10,000
30002	Sinsenveien 34	25,000	8,000
30003	Prof. Hansens vei 22	45,000	15,000
30004	Utsikten	50,000	17,500
30005	Knatten 22	30,000	10,000
30006	Pilestredet 35	20,000	700

Goruddalen 1, 0453 Oslo

Annonse ID	30001	Depositum	kr. 30,000,-	Pris pr mnd	kr. 10,000,-
Byggeår	1970	Boareal	75 m2	Tilgjengelig fra:	29-06-2014
Etasje	8	Bodareal	10 m2	Balkong	Ingen balkong
Har heis?	Ja	Har garasje?	Nei	Fellesvaskeri?	Ja

Beskrivelse av bolig
Flott leilighet, solvendt.

Selgers krav til leietaker
Ikke lov å røyke i boligen.

Mangler Bilde

Send forespørsel

Figur 4.4: Visningspanel.

4.2.4 Forespørsel/søknad

Trykk på “Send forespørsel” knappen nede i venstre hjørne av programmet (alternativt CTRL-F), for sende inn din søknad på valgt bolig. Du vil da bli spurt om å akseptere evt. spesielle vilkår (figur 4.5) for den valgte boligen.

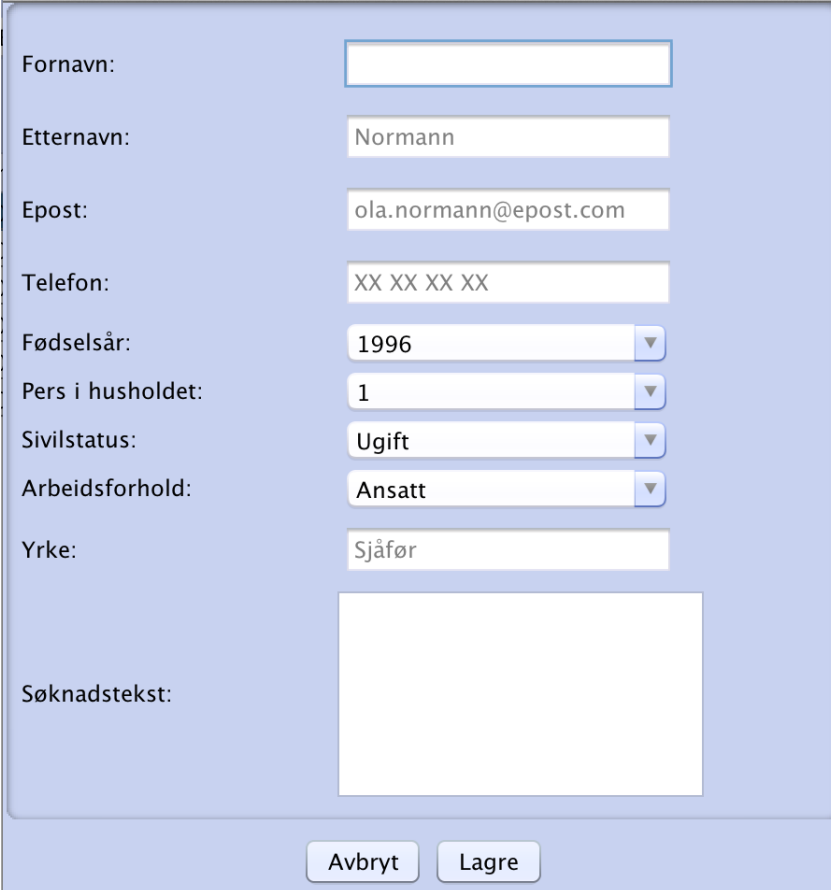
Vennligst aksepter vilkårene:

Ikke lov å røyke i boligen.

Avbryt Aksepter

Figur 4.5: Forespørsel og krav fra utleier.

Etter å evt. ha akseptert vilkårene for den aktuelle boligen, så får en opp et nytt vindu med mulighet for å skrive inn personalia mm. og sende inn søknad på boligen, som vist på figur 4.6.



A registration form for tenants, displayed in a light blue window. The form contains several input fields and dropdown menus. The fields are labeled on the left and the input area is on the right. The labels are: Fornavn, Etternavn, Epost, Telefon, Fødselsår, Pers i husholdet, Sivilstatus, Arbeidsforhold, Yrke, and Søknadstekst. The input fields contain the following values: Fornavn (empty), Etternavn (Normann), Epost (ola.normann@epost.com), Telefon (XX XX XX XX), Fødselsår (1996), Pers i husholdet (1), Sivilstatus (Ugift), Arbeidsforhold (Ansatt), Yrke (Sjåfør), and Søknadstekst (empty text area). At the bottom of the form are two buttons: Avbryt and Lagre.

Fornavn:	<input type="text"/>
Etternavn:	<input type="text" value="Normann"/>
Epost:	<input type="text" value="ola.normann@epost.com"/>
Telefon:	<input type="text" value="XX XX XX XX"/>
Fødselsår:	<input type="text" value="1996"/>
Pers i husholdet:	<input type="text" value="1"/>
Sivilstatus:	<input type="text" value="Ugift"/>
Arbeidsforhold:	<input type="text" value="Ansatt"/>
Yrke:	<input type="text" value="Sjåfør"/>
Søknadstekst:	<div></div>

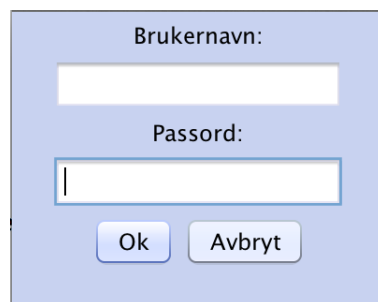
Avbryt Lagre

Figur 4.6: Registrering for leietaker.

4.3 Megler/administrasjon

4.3.1 Pålogging

Ved å velge “Megler”-arkfanen, så vil du først bli spurt om å logge deg inn som vist på figur 4.7. Tast inn ditt brukernavn og passord, og trykk “Enter” for å logge inn, eller evt. ved å trykke på “Ok” knappen. Velger du “Avbryt” vil du komme tilbake til Annonse-arkfanen igjen.

A light blue rectangular dialog box with a thin black border. It contains two text input fields. The first field is preceded by the label "Brukernavn:" and the second by "Passord:". Below the second field are two buttons: "Ok" and "Avbryt".

Figur 4.7: Pålogging for megler.

4.3.2 Menyer

I den øverste delen av programmet som vist på figur 4.8, så har du en knappegruppe som er ansvarlig for administrasjon, mer spesifikt funksjonalitet for å opprette ny utleier, ny bolig, ny annonse, og ny kontrakt.

Hvilke av disse funksjonene / knappene som er mulig å bruke, er avhengig av hvilken type objekt du har valgt i i vinduet for søkeresultater. Mer spesifikt:

Ny Utleier - Kan brukes når som helst. CTRL-U

Ny Bolig - Kan brukes etter å ha søkt og valgt blant utleiere. CTRL-B

Ny Annonse - Kan brukes etter å ha søkt og valgt blant boliger. CTRL-A

Ny Kontrakt - Kan brukes etter å ha søkt og valgt blant søknader. CTRL-K

4.3.3 Søkepanel

I øverste delen av programmet som vist på figur 4.8, så har du mulighet til å søke blant boligsøknader, annonser, boliger, utleiere, leietakere, og eksisterende kontrakter. Du må velge en kategori på venstre siden av søkefeltet for å kunne ta i bruk søkefunksjonaliteten, ved hjelp av å trykke “Enter”, eller ved å enkelt og greit trykke på “søkeknappen”.

BoligID	Adresse	Annonisert	Utleid
20001	Ivar Aasens vei 23	Nei	Nei
20002	Ivar Aasens vei 24	Nei	Nei
20003	Prof. Hansens vei 22	Ja	Nei
20004	Utsikten	Ja	Nei
20005	Eneboligstrøket 44	Nei	Nei
20006	Gladengveien 15A	Nei	Nei
20007	Sinsenveien 34	Ja	Nei
20008	Groruddalen 1	Ja	Nei
20009	Knatten 22	Ja	Nei
20010	Pilestredet 35	Ja	Nei

Figur 4.8: Søkepanel.

4.3.4 Resultattabell

Dette er venstre del av programvinduet, markert i rødt på figur 4.9. Her vil resultater av et evt. utført søk vises.

En har mulighet til å sortere på diverse kriterier øverst i resultatvinduet. Du kan bla igjennom annonser ved å bruke pil opp, eller pil ned knappene, eller alternativt bruke “fram og tilbake” knappene nederst til høyre i programvinduet.

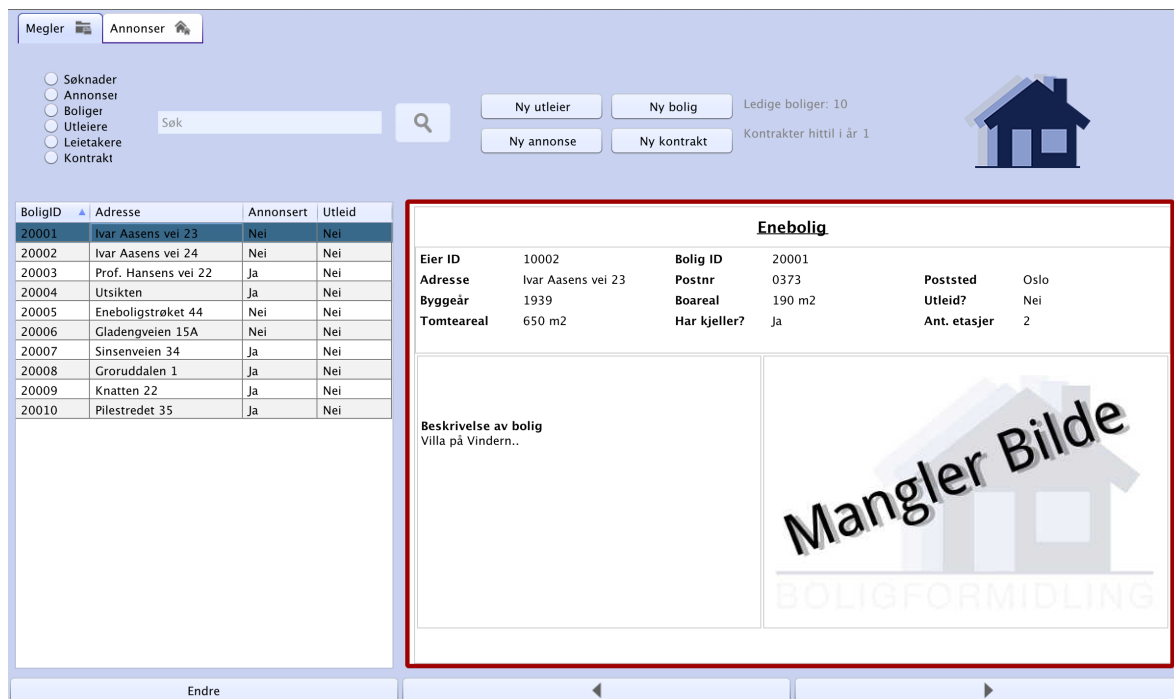
The screenshot shows a web application interface for real estate. At the top, there are tabs for 'Megler' and 'Annonser'. Below the tabs, there are radio buttons for 'Søknader', 'Annonser', 'Boliger', 'Utleiere', 'Leietakere', and 'Kontrakt'. A search bar with the text 'Søk' and a magnifying glass icon is present. To the right of the search bar, there are buttons for 'Ny utleier', 'Ny bolig', 'Ny annonse', and 'Ny kontrakt'. Further right, there are statistics: 'Ledige boliger: 10' and 'Kontrakter hittil i år 0'. A house icon is also visible. The main content area is divided into two parts. The left part contains a table with search results, which is highlighted with a red border. The right part is a large empty space. At the bottom, there are navigation buttons: 'Endre', a left arrow, and a right arrow.

BoligID	Adresse	Annonisert	Utleid
20001	Ivar Aasens vei 23	Nei	Nei
20002	Ivar Aasens vei 24	Nei	Nei
20003	Prof. Hansens vei 22	Ja	Nei
20004	Utsikten	Ja	Nei
20005	Eneboligstrøket 44	Nei	Nei
20006	Gladengveien 15A	Nei	Nei
20007	Sinsenveien 34	Ja	Nei
20008	Groruddalen 1	Ja	Nei
20009	Knatten 22	Ja	Nei
20010	Pilestredet 35	Ja	Nei

Figur 4.9: Resultattabell.

4.3.5 Visningspanel

Her vil detaljer rundt valgt objekt (i resultattabellen) vises, markert i rødt på figur 4.10.



Megler **Annonser**

☐ Søknader
☐ Annonser
☐ Boliger
☐ Utleiere
☐ Leietakere
☐ Kontrakt

Søk

Ny utleier Ny bolig
Ny annonse Ny kontrakt

Ledige boliger: 10
Kontrakter hittil i år 1

BoligID	Adresse	Annonisert	Utleid
20001	Ivar Aasens vei 23	Nei	Nei
20002	Ivar Aasens vei 24	Nei	Nei
20003	Prof. Hansens vei 22	Ja	Nei
20004	Utsikten	Ja	Nei
20005	Eneboligstrøket 44	Nei	Nei
20006	Gladengveien 15A	Nei	Nei
20007	Sinsenveien 34	Ja	Nei
20008	Groruddalen 1	Ja	Nei
20009	Knatten 22	Ja	Nei
20010	Pilestredet 35	Ja	Nei

Enebolig

Eier ID	10002	Bolig ID	20001
Adresse	Ivar Aasens vei 23	Postnr	0373
Byggeår	1939	Boareal	190 m2
Tomteareal	650 m2	Har kjeller?	Ja
		Poststed	Oslo
		Utleid?	Nei
		Ant. etasjer	2

Beskrivelse av bolig
Villa på Vindern..

Mangler Bilde

BOLIGFORMIDLING

Endre

Figur 4.10: Visningspanel.

4.3.6 Utleieradministrasjon

Ved å søke blant utleiere, velge en person i listen, og trykke på “Endre” knappen nederst i venstre hjørne av programvinduet, så har en mulighet til å gjøre endringer i informasjon lagret om vedkommende, som vist på figur 4.11.



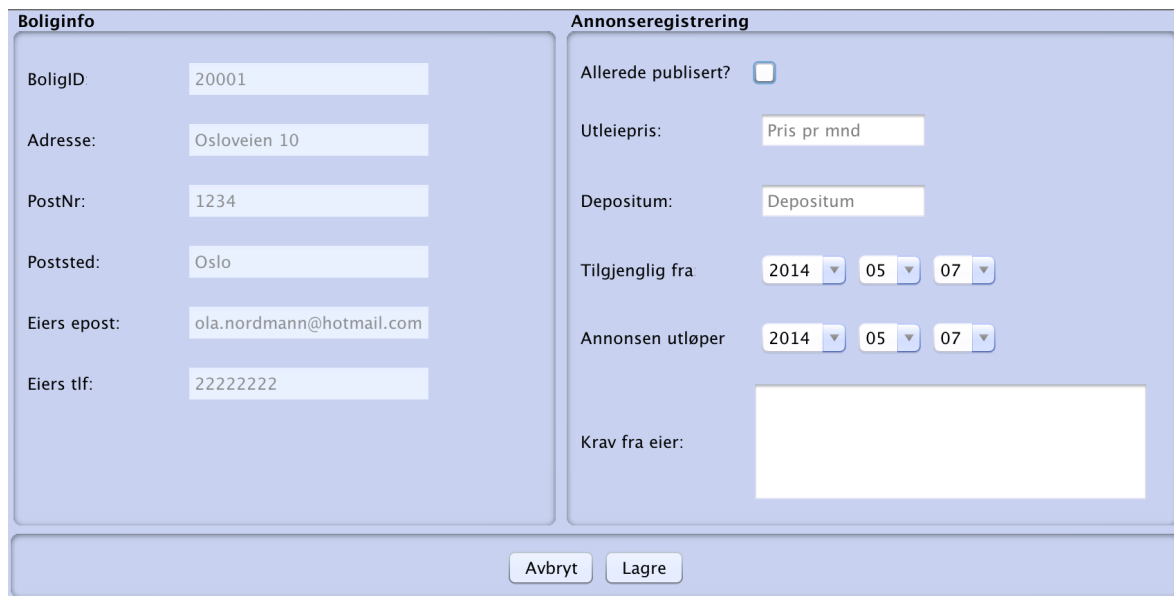
Fornavn:	Petter
Etternavn:	Stordalen
Epost:	pstordalen@yahoo.com
Telefon:	23904532
Er representant:	<input type="checkbox"/>
Representerer:	Navn på representat

Avbryt Lagre

Figur 4.11: Utleieradministrasjon.

4.3.7 Annonseadministrasjon

Ved å søke blant annonser, velge en annonse i listen, og trykke på “Endre” knappen nedest i venstre hjørne av programvinduet, så har en mulighet til å gjøre endringer i informasjon lagret om annonsen, som vist på figur 4.12.



Boliginfo		Annonseregistrering	
BoligID	20001	Allerede publisert?	<input type="checkbox"/>
Adresse:	Osloveien 10	Utleiepris:	Pris pr mnd
PostNr:	1234	Depositum:	Depositum
Poststed:	Oslo	Tilgjengelig fra	2014 05 07
Eiers epost:	ola.nordmann@hotmail.com	Annsen utløper	2014 05 07
Eiers tlf:	22222222	Krav fra eier:	
		Avbryt Lagre	

Figur 4.12: Annonseadministrasjon.

4.3.8 Boligadministrasjon

Ved å søke blant boliger, velge en bolig i listen, og trykke på “Endre” knappen nederst i venstre hjørne av programvinduet, så har en mulighet til å gjøre endringer i informasjon lagret om boligen, som vist på figur 4.13.

Leilighet ☐ Enebolig ☐

Felles

Eier ID: 10002

Megler ID: 10001

Adresse: Ivar Aasens vei 23

Postnummer: 0373

Poststed: Oslo

Boareal: 190

Byggeår: 1939

Kan leies fra: 2014 07 01

Utleid: ☐

Beskrivelse: Villa på Vindern..

Antall bilder: 0

Bilder:

Leilighet

Etasje nr: XX

Balkong areal: XX

Areal bod: XX

Heis: ☐

Garasje: ☐

Fellesvaskeri: ☐

Enebolig

Antall etasjer: 2

Tomt areal: 650

Kjeller: ☒

Avbryt Lagre

Figur 4.13: Boligadministrasjon.

4.3.9 Kontraktadministrasjon

Ved å søke etter innkommende søknader, velge en søknad i listen, og trykke på “Ny kontrakt” knappen øverst i programvinduet, så har en mulighet til å godta, eller avslå den aktuelle søknaden. Alternativt kan en høyreklikke på den aktuelle søknaden i listen, og velge godta/avslå derfra som vist på figur 4.14. Er det flere søknader på en bolig, så vil de andre bli avslått ved en evt. godkjennelse av en søknad.

Annon...	Søkers epost	Behan...	Godkjent
30004	ola.nordmann@gmail	...	Nei
30001	petterlysne@hotmail.	Aksepter søknad Avvis søknad	a

Figur 4.14: Kontraktadministrasjon.

4.3.10 Sletting

Ved å høyreklikke på et objekt i resultatlisten, så har man mulighet til å slette det valgte objektet. Alternativt kan “Delete” knappen brukes.

Det er imidlertid noen krav for å få slettet bestemte objekter. Mer spesifikt:

Slette utleiere - En kan ikke slette utleiere med eksisterende registrerte boliger. Boligene må i såfall slettes først. Boliger som er utleid kan heller ikke slettes før de evt. ikke lenger er utleid.

4.3.11 Hurtigtaster/hotkeys

For avanserte brukere, så kan hurtigtaster brukes for å oppnå høyere effektivitet, og kjappere saksbehandling. Følgende hurtigtaster kan brukes:

Generelt

CTRL-Tab - Skifter mellom annonse og megler arkfane / tab.

Enter/return - Utfører et søk med gjeldende valgte søkekriterier.

Pil ned - Blar nedover evt. søketreff.

Pil opp - Blar oppover evt. søketreff.

Kapittel 5

Testrapport

5.1 Testing utført

5.1.1 Testing underveis

Vi kom sent i gang med strukturert testing av det ferdige produktet. Helt siden vi bare hadde en datastruktur har vi hatt en metode som legger inn dummydata i registerne, så de har alltid vært solide. Når vi kom så langt at vi begynte å implementere programlogikk og brukergrensesnitt ble behovet for tilbakemelding om eventuelle **exceptions** og gale innstillinger en nødvendighet. Stort sett har vi bare brukt terminal og `System.out.println` for å gi beskjed om status på forskjellige variabler, feks før og etter en instruksjon er kjørt, eller inne i `try/catch`-blokker.

5.1.2 Funksjonstest av ferdig program

Etter overgang til **Nimbus Look and Feel** har brukeropplevelsen i Linux og Windows blitt nærmest lik, høyreklikkmenyen fungerer bedre i Windows. Det er gjort to identiske tester for å teste funksjonaliteten i Linux og Windows. I Linux kjørt programmet fra utviklermiljøet til Netbeans, mens i Windows ble det kjørt fra `jar`-fil slik som sensor vil gjøre. Testing i Linux ble utført først, og de feil som ble oppdaget ble rettet underveis. Det var derfor ikke mye feil å rapportere under testing i Windows.

Resultat av gjennomførte tester presenteres i tabell 5.1 og tabell 5.2.

Tabell 5.1: Testrapport ServiciosDeVivienda – Linux (Ubuntu 12.04)

Hva er testet	Resultat	Status
Registrer Utleier	Registrering av ny utleier får fint. RegEx fungerer bra.	OK
	Tastatursnarveien CTRL-u fungerer fint, og tabulator mellom feltene fungerer.	OK
Endre Utleier	Endre informasjon om en utleier fungerer bra.	OK
Slette Utleier	Det fungerer bra. Om en utleier har en bolig registrert kan utleier ikke slettet	OK
Registrer Bolig	Registrering av leilighet går fint. RegEx på Bod og Balkong må fikses.	Alt OK. Feilene er fikset.
	Legge inn bilder under registrering går fint. Det samme gjør å oppdatere bolig senere med bilder.	OK
Endre Bolig	Endring av boligobjektet går for så vidt fint, men henter ikke opp Areal bod.	Feilene er fikset
Slette bolig	Fungerer fint.	OK
Registrer Annonse	Registrert annonse via Ny annonse. Det gikk fint. Boligen i tabellen registrerer at den nå er annonsert.	OK
Endre Annonse	Endre Annonse oppdaterer ikke.	Feilene er fikset.
	Bolig som har vært annonsert kan tas av «nett» og aktiveres igjen. Fungerer bra.	OK
Opprette Søknad	Fungerer fint.	OK
Akseptere Søknad	Fungerer fint. Søknader på samme annonse markeres som behandlet og avvist, blir «grået ut» og lagt nederst i tabellen. Har en person allerede fått akseptert sin søknad på en annonse, kan han ikke få godkjent andre søknader.	OK

Tabell 5.2: Testrapport ServiciosDeVivienda - Windows 8.1

Hva er testet	Resultat	Status
Registrer Utleier	Fungerer fint	OK
Endre Utleier	Fungerer fint	OK
Slette Utleier	Fungerer fint	OK
Registrer Bolig	Fungerer fint	OK
Endre Bolig	Fungerer fint	OK
Slette bolig	Fungerer fint	OK
Registrer Annonse	Fungerer fint	OK
Endre Annonse	Fungerer fint	OK
Opprette Søknad	Fungerer fint	OK
Akseptere Søknad	Fungerer fint	OK
Høyreklikkmeny	Fungerer ikke. Fix: Har laget egen MouseEvent for Windows	Fikset. OK

5.2 Begrensninger

5.2.1 Søk

Søkeklassen for megler (avsnitt 3.8.1, side 44) fungerer meget bra på de datamengder som programmet er testet på. På sikt burde det gjennomføres en rekke forbedringer i søket. Foreløpig itererer søket gjennom hele datasettet, selv om den skal returnere et tom søk. Søket er også alt for generelt, det medfører at søket resulterer alt for mange treff. For eksempel det er fullt mulig å søke med en enkel bokstav, hvis brukeren skriver «a» i søkefeltet blir alle objekter som inneholder den bokstaven i sine datafelt returnert. Eventuelle forbedringer som kan legges til kan være at brukeren kan spesifisere enkelte datafelt som man ønsker å søke i, i tillegg til ønsket register. For eksempel dersom man ønsker å søke i personregister på alle som har fornavn Ola, kan brukeren skrive søket med følgende «syntaks»: `fornavn:Ola`. Slik søk vil da begrense søket til de spesifikke datafeltet.

5.2.2 Generell brukeropplevelse

Normalt sett vil et program ha mer funksjonalitet for sluttbruker enn det dette prosjektet tilbyr. Mangel av tid er hovedgrunnen til det, men også at vi ikke hadde et sterkere fokus på det spesifikke en sluttbruker ofte ønsker å få gjort helt fra starten. Det var for så vidt et valg vi bevisst tok, om å fokusere på det tekniske, men skulle vi begynt på nytt er det klart at toppanel for både megler og annonse-visning ville sett litt annerledes ut. Programmet burde også kunne kjøres i fullskjermvisning", noe vi valgte å ikke tillate da forskjellige skjermoppløsninger ville ødelegge layout for både tabell og visningsområde for objektene.

5.3 Kjente feil (Bugs)

5.3.1 Feil i fritekstsøk

Det er funnet én åpenbar bug eller mangel som vi er klar over, men ikke kan fikse da det vil ødelegge datasettet vårt. Det vil si at vi kan ikke gjøre endringer i objektene for å fikse feilen uten å ødelegge datasettet med eksempeldata. Feilen består i at når en benytter fritekstsøket i meglerpanelet og søker på en Utleier eller Leietaker, så får man to resultater per person.

Grunnen til dette er at klassen `Person.java` implementerer interfacet `Searchable.java` der metoden `toSearch()` definerer hva som skal være søkbart. Det i seg selv er greit, men subclassene til `Person`, `Leietaker`, `Utleier` og `Megler` arver denne metoden og dermed returnerer de også samme objekt som superklassen.

FIX: For å fikse dette midlertidig er søkerutinens datasett endret fra `ArrayList` til `HashSet`. Dette for å hindre at søkeresultatet som sendes til tabellen ikke vil inneholde mer en én referanse til samme objekt. Permanent fix ville vært å la subclassene til `Person.java` override superklassens `toSearch()`-metode, og bare inkludere unike felt for de objektene.

5.4 Forbedringer

Bedre implementasjon av annonsesøk

Vi fulgte oppgaveteksten ganske nøye fra begynnelsen av, og betalte kanskje litt for det mot slutten da vi så at søkemulighetene for boligsøker var litt begrenset. Dette er likevel en veldig tidkrevende implementasjon som ville krevd mer tid enn vi har hatt tilgjengelig.

Utvidet funksjonalitet i tabellen

Tabellens implementasjon er veldig vellykket, men mulighetene en tabell tilbyr er også enorme. Vi valgt å bare ha 4 kolonner tilgjengelig for brukeren, men en kan se for seg at man kunne skjule "output-vinduet" og få en mye bredere tabell, som da også ville gitt bedre mulighet for å kunne editere objekter direkte i tabellen. En kunne også se for seg filtrering direkte i tabellen i stedet for eller i tillegg til dagens søkefunksjon.

Generell kommunikasjon i programmet

Vi ga oss selv relativt store utfordringer da vi valgt MVC-arkitektur gjennom hele programmet samtidig som vi ønsket en tabell som responderer i det man gjør endringer. Vi har ikke hatt tid til å få tabellen til å respondere på absolutt alle endringer fra alle forskjellige vinkler. Med vinkler menes hvor man trykker for å få utført en operasjon. I det man klikker på Ny Utleier-knappen i topppanelet så utføres det fra en annen plass enn om en høyreklikker i tabellen og velger ny person.

Om mulig burde det lages en «kommunikasjonssentral» som alltid kjenner til og oppfatter alle hendelser. I det man er i et registreringsvindu så har man ikke kontakt med hovedprogrammet. Vi har løst dette så godt som mulig ved å sendes oppdateringshendelsen via et **interface**, som nok ville vært behov for uansett, men kanskje en helhetlig kommunikasjonsstruktur gjennom hele programmet ville vært lettere å jobbe med når en får ytterligere kompleksitet og utfordringer ettersom programmet utvikles.

Bruk av SQL som datakilde

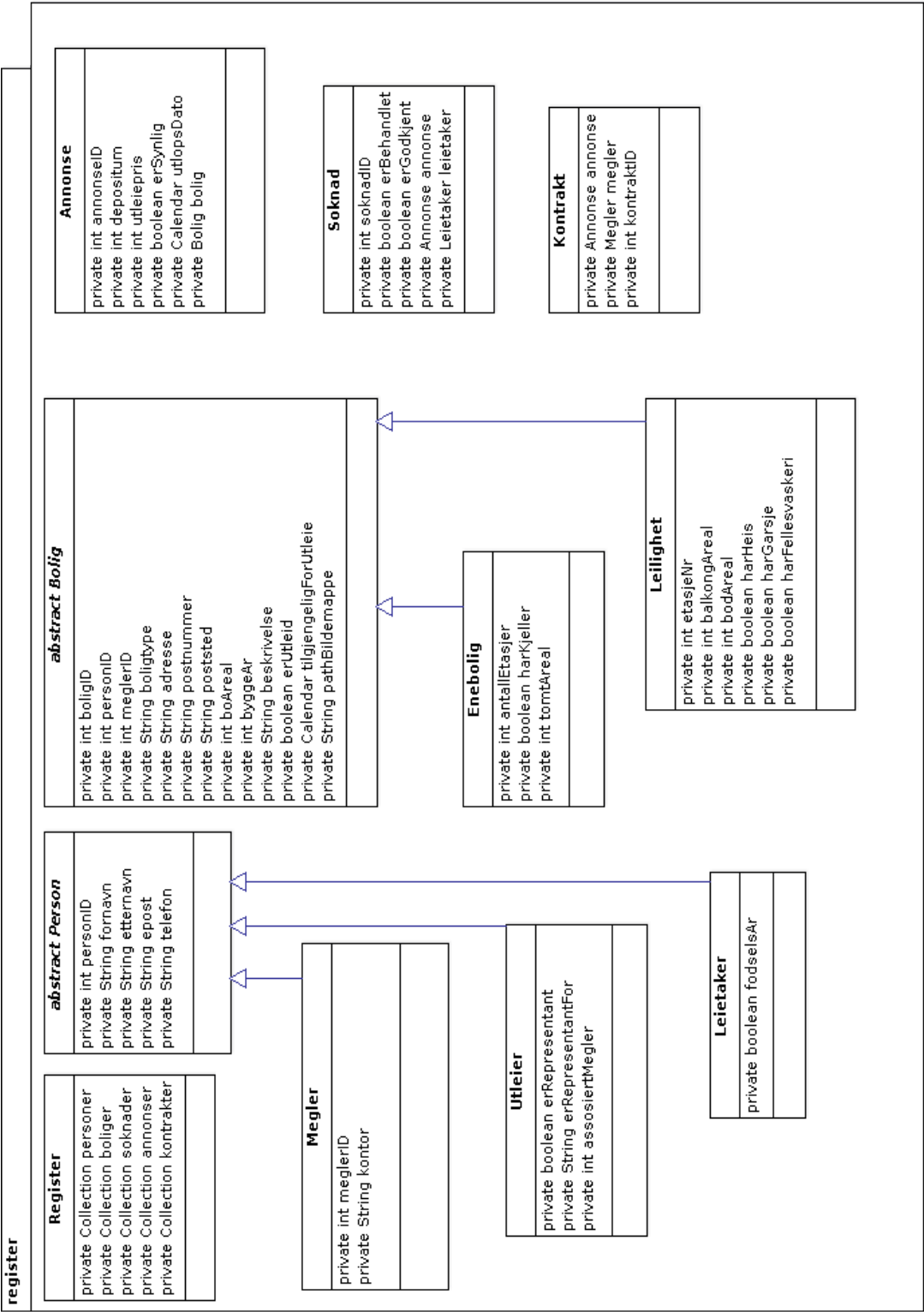
Denne oppgaven ville vært ideell for lagring i SQL i stedet for å serialisere dataene. Det ville vært normalt med mye spørringer på tvers tabellene for å presentere statistikk, samt å lettere kunne skreddersy søkerutiner.

Statistikk

Tidsmangel gjorde at det ikke ble vektlagt noe utvidet statistikkpanel. Dette var noe vi ønsket å få til, men det vil være å anse som første prioritert ved eventuell videreutvikling av programmet.

Tillegg A

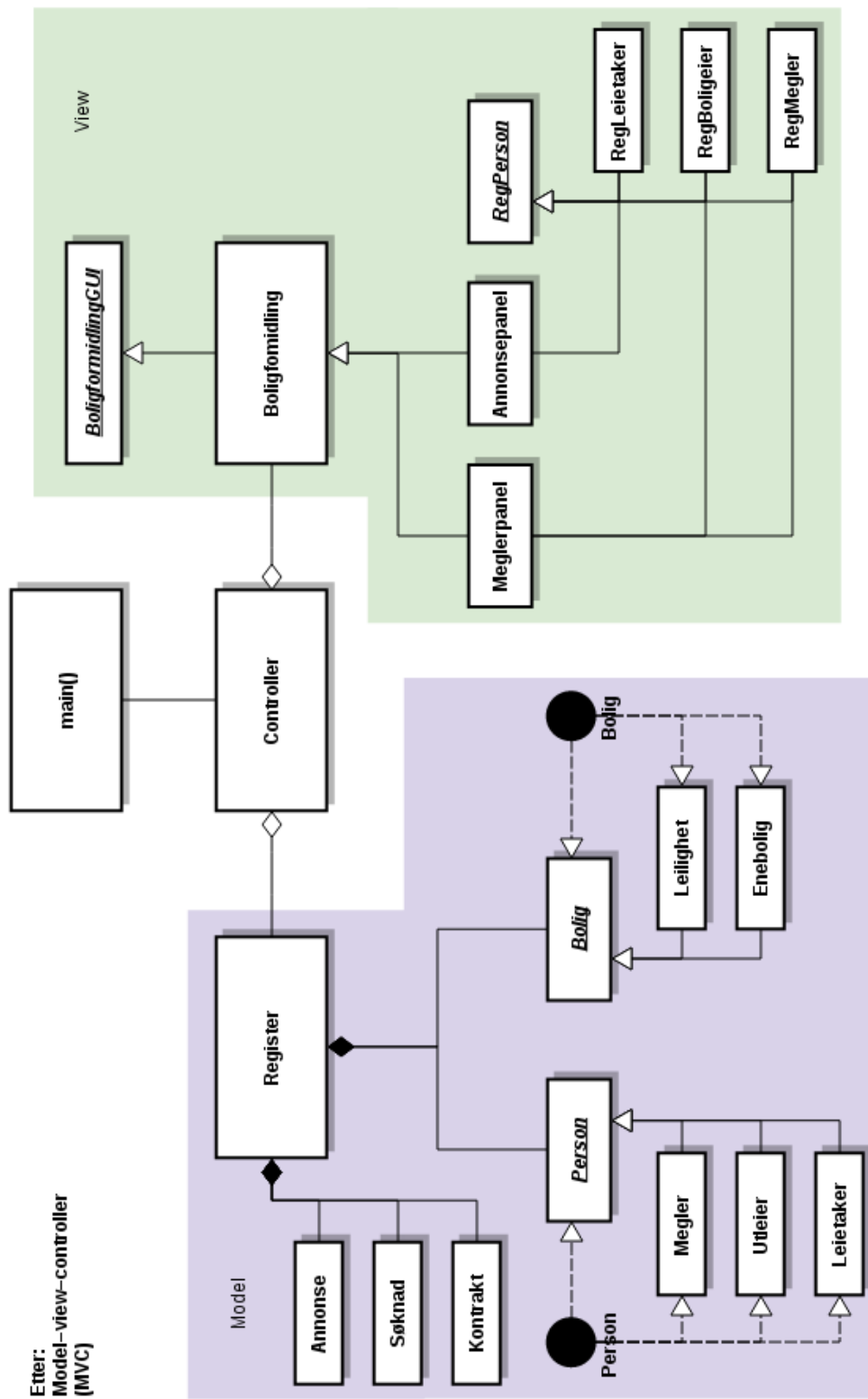
Diagram



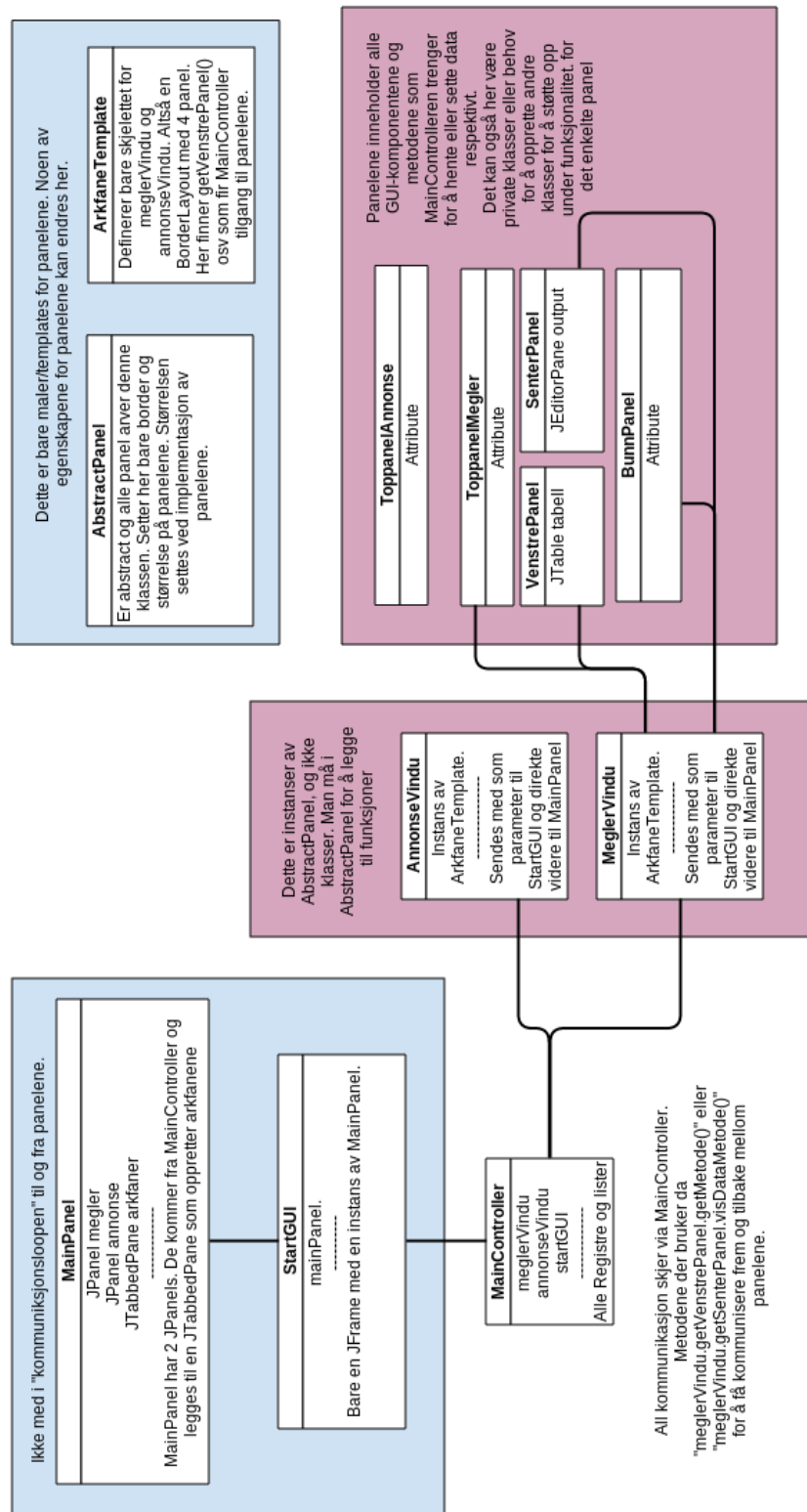
Figur A.1: Innledende UML diagram, brukt for generering av grunnleggende klasser.



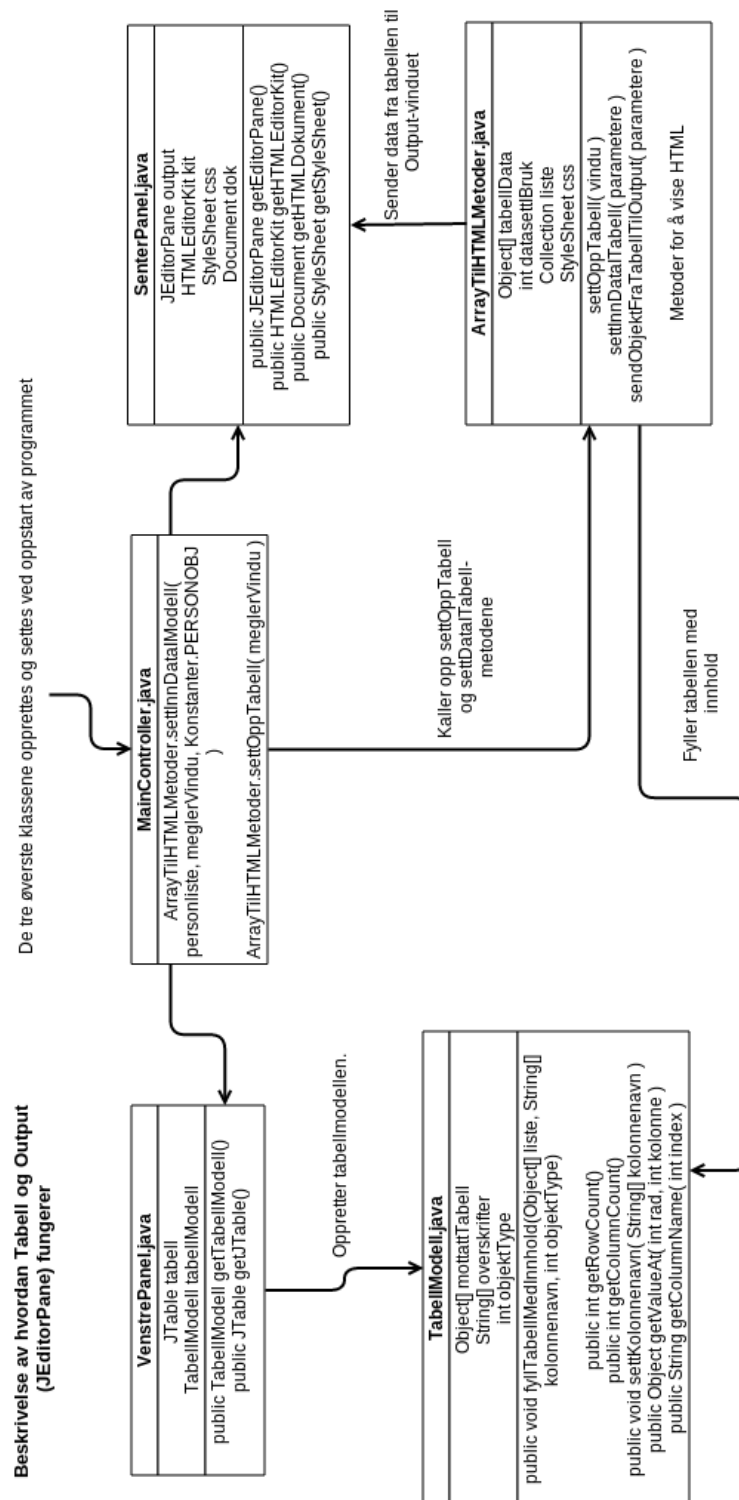
Figur A.2: Flyt diagram over mulig user case som kan foretas i brukergrensesnittet.



Figur A.3: Innledende diagram over planlagt MVC arkitektur i programmet.



Figur A.4: Flytdiagram over kontroller og GUI klasser.

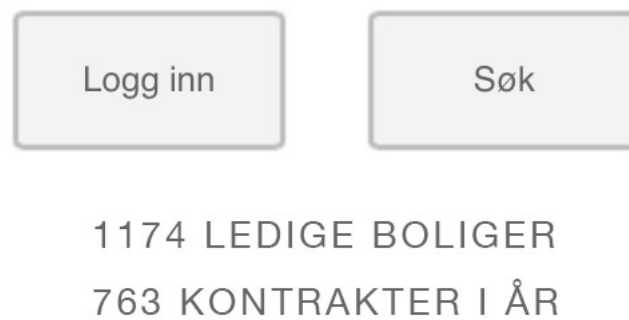


Figur A.5: Flytdiagram over tabellmodell og output.

Tillegg B

Fremtaging av GUI

B.1 Revisjon 1



Figur B.1: Fremtaging: Velkomstbilde

The GUI for 'Boligsøk' features a search interface at the top. It includes three dropdown menus for 'Antall rom', 'Boligtype', and 'Fylke:'. Below these are three input fields for 'Areal min:', 'Areal max:', and 'Max utleiepris:'. A 'Finn boliger' button is on the left, and a 'Søk valgt bolig' button is on the right. The main area is a large, empty rectangular box for displaying search results.

Figur B.2: Fremtaging: Boligsøk

The GUI for 'Meglersøk' has a search interface at the top with five input fields: 'Fornavn', 'Etternavn', 'FødselsNr/dato', 'Adresse', and 'KontraktNr'. Below these fields are six buttons: 'Finn Bolig', 'Finn Person', 'Finn Kontrakt', 'Slett Bolig', 'Slett Person', and 'Slett Kontrakt'. The main area is a large, empty rectangular box for displaying search results.

Figur B.3: Fremtaging: Meglersøk

The screenshot shows a web application interface with three tabs at the top: "Registrer", "Søk", and "Saksbehandling". The "Registrer" tab is active. Below the tabs is a registration form with the following elements:

- Four input fields in the first row: "Fornavn", "Epost", "PostNr", and "Byggeår".
- Four input fields in the second row: "Etternavn", "TelefonNr", "Poststed", and "Utleiepris (mnd)".
- Four input fields in the third row: "PersonNr", "Adresse", "Areal", and "Antall rom" (which has a dropdown arrow).
- A checkbox labeled "Hus/rekkehus" which is checked.
- A section titled "Andre opplysninger" containing a large text area and a date selector with fields for "mm", "dd", and "yyyy", along with a calendar icon.
- A "Registrer" button centered below the "Andre opplysninger" section.
- A checkbox labeled "Leilighet" which is unchecked.
- A section on the right containing a dropdown for "Etasje", a checkbox for "Heis", and a checkbox for "Balkong".
- A section on the left containing a dropdown for "Etasjer", a text field for "Tomtstørrelse", and a checkbox for "Kjeller".

Figur B.4: Fremtaging: Megler GUI

This screenshot shows the same registration form as Figure B.4, but with different states for the checkboxes:

- The "Hus/rekkehus" checkbox is checked.
- The "Leilighet" checkbox is unchecked.
- The "Kjeller" checkbox is unchecked.
- The "Heis" checkbox is unchecked.
- The "Balkong" checkbox is unchecked.

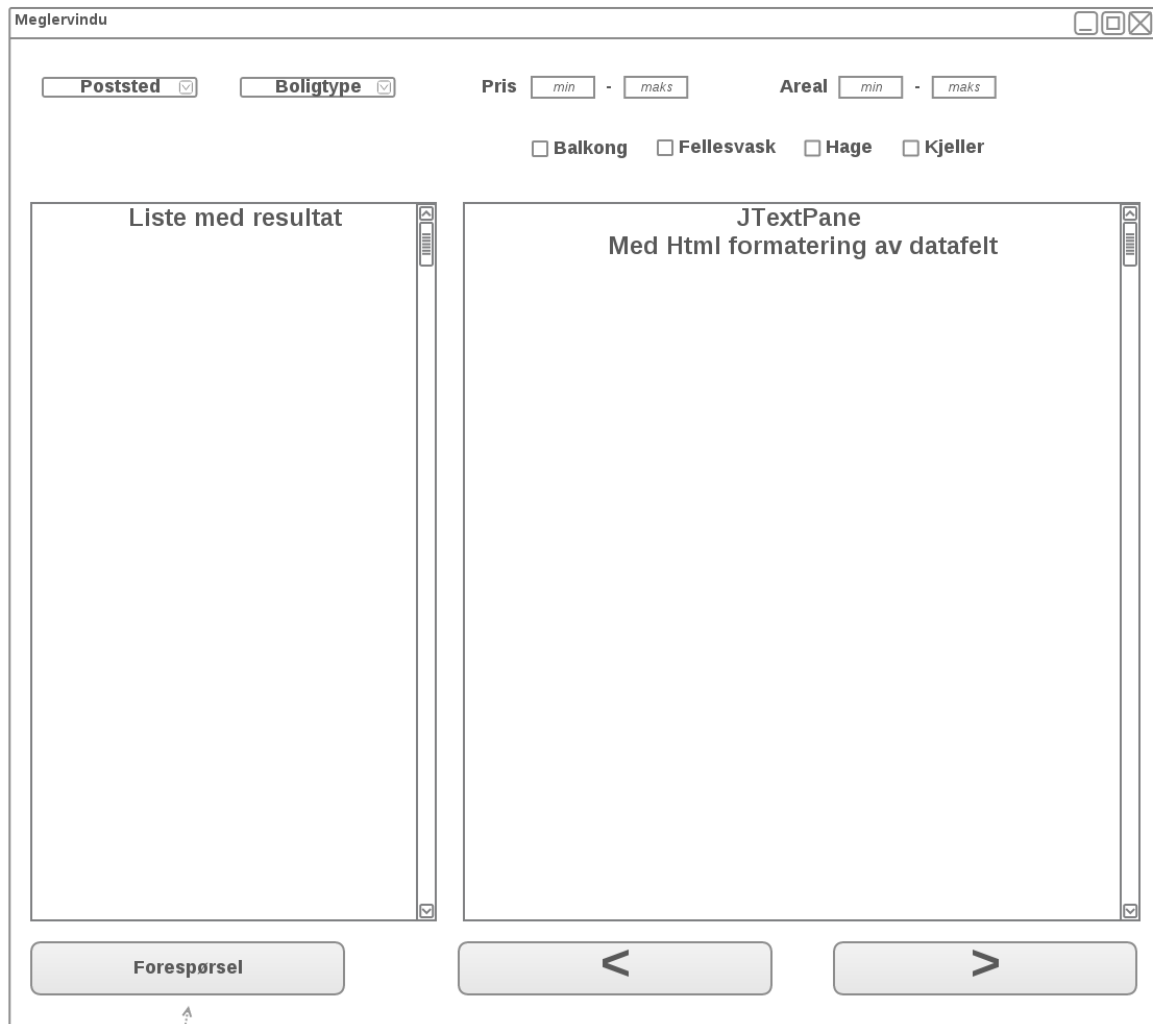
All other fields and the layout are identical to Figure B.4.

Figur B.5: Fremtaging: Boligregistrering



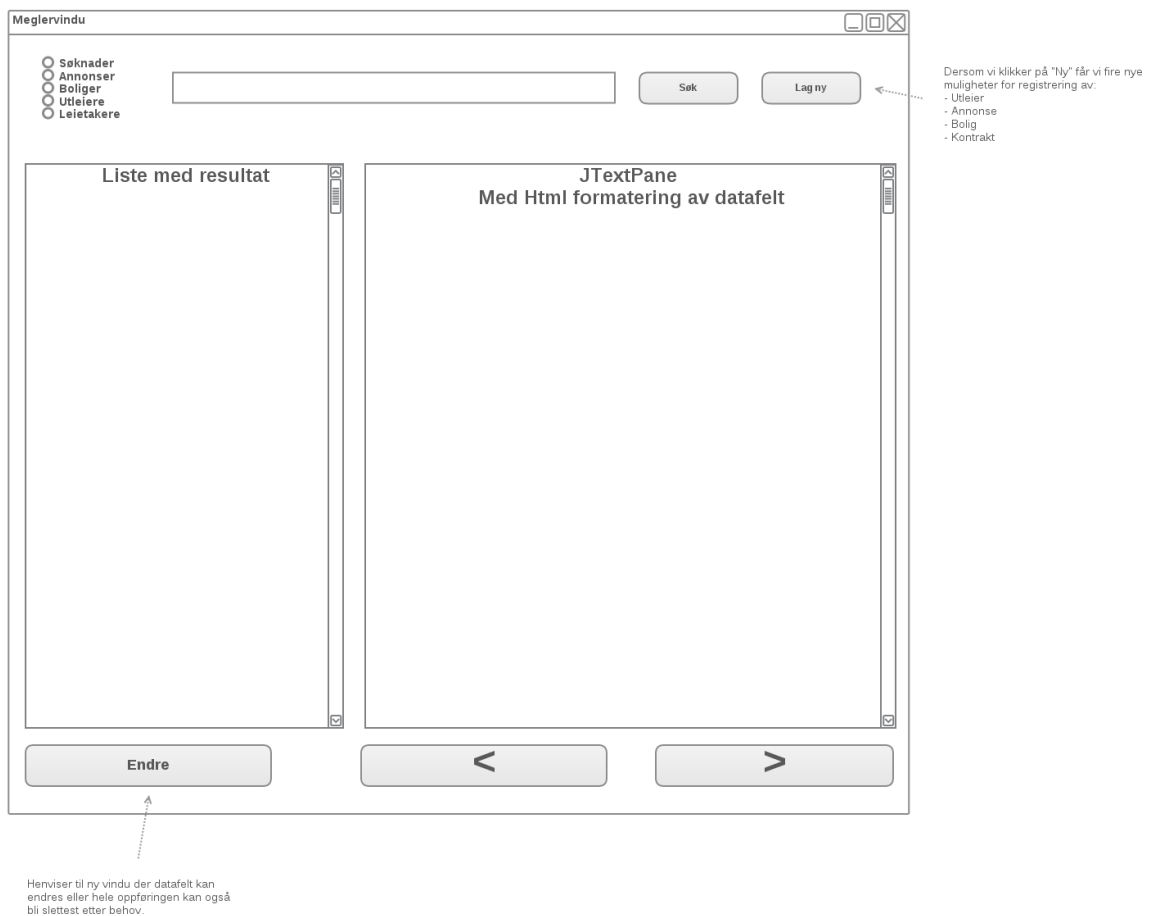
Figur B.6: Fremtaging: Saksbehandling

B.2 Revisjon 2



Henviser til ny vindu der datafelt kan endres eller hele oppføringen kan også bli slettet etter behov.

Figur B.7: Fremtaging 2: Søkervindu



Figur B.8: Fremtaging 2: Meglervindu