INF 1050
OBLIGATORISK OPPGAVE 3

GUI-PROTOTYPING OG DATABASER

9 sider

LEVERINGSFRIST: Fredag 4/5-2009, kl. 16:00

Evaluering: Bestått/Ikke bestått. Du må ha bestått denne obligatoriske oppgaven for å gå opp til eksamen. I bedømmingen av denne oppgaven, vil det legges vekt på forståelsen av pensum og verktøyet Genova som du viser gjennom innleveringen.

Formål: I denne oppgaven skal du øve deg i å generere og endre GUI-prototyper, samt lage database-skjema på grunnlag av en UML-modell.

Forutsetninger: Oppgaveteksten vil uten videre bruke begreper fra forelesninger, lærebøker og anbefalt lesestoff.

Leveranser: Du skal levere en individuell besvarelse. Du skal levere en fil som inneholder bilder av dialogvinduene du skal generere. Fila skal hete **<fornavn_etternavn>OBLIG3.pdf**. Du skal også levere to filer som inneholder databaseskjemaer som du skal generere. Disse filene skal hete **<fornavn_etternavn>OBLIG3.sql** og **<fornavn_etternavn>OBLIG3_b.sql**.

Levering: Filene skal sendes til **inf1050-<gruppenr>@ifi.uio.no**, der **<gruppenr>** er nummeret på gruppa du går på. I subject-feltet skal det stå **INF1050-oblig3**, slik at innleveringen kan registreres automatisk. Du bør dessuten sende eposten fra din Ifi/UiO-konto for å unngå at innleveringen din stanses av spamfiltre. Leveringsfristen er absolutt. Gruppelærer har intet mandat til å gi utsettelse (annet enn ved sykdom ved egenmelding). Eventuelle utsettelser må tas opp med studieadministrasjonen.

Krav om autentisitet og regler for obligatoriske oppgaver: Se skrivene fra instituttet som var inkludert med Oblig 1.

PROBLEMBESKRIVELSE:

Du og NoChoice har nå blitt enige om det overordnete designet til systemet, samt andre relevante kontraktsforhold. Blant annet har dere blitt enige om at det skal brukes en ansvarsdrevet designprosess i hver iterasjon, og at man skal benytte en trelags-arkitektur med kant-, kontroll- og entitetsobjekter, der sistnevnte lagres i en relasjonsdatabase som kunden (NoChoice) anskaffer. Disse forholdene kan du anta at dere har stadfestet i punktene under C2 i PS2000. Videre har dere bestemt at en del av kontrollpunktene til hver iterasjon (se figuren f.eks. under C1 i PS2000), skal bestå av at kunden (NoChoice) får presentert en GUI- og Web-prototyper til godkjenning hver gang ny funksjonalitet designes ferdig, dersom denne funksjonaliteten medfører endringer i kantobjektene. (Iterasjoner der kontrollpunktet er godkjenning av prototyper har ikke nødvendigvis testing av kode.) Dette kan du anta at dere har stadfestet i punktene C4.3.1 og C4.3.2 i PS2000. (GUI=Graphical User Interface=grafisk brukergrensesnitt=skjermbilde.)

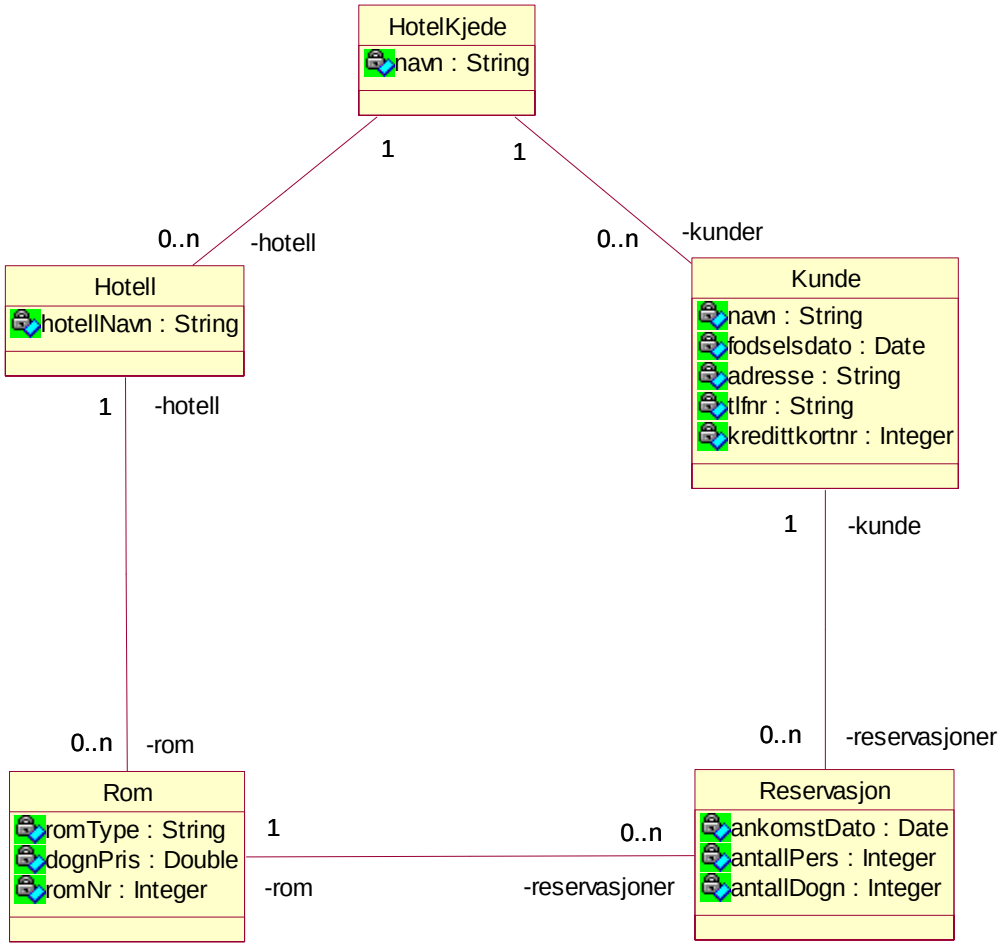
Oppgave 1 GUI: Til GUI-prototyping har du hyret inn ekstern hjelp fra Esito AS. Esito har laget et GUI-prototypingsverktøy som heter Genova. (Genova kan for øvrig også automatisk generere kjørbar kode fra UML-diagrammer samt generere databaseskjemaer) Første iterasjon i prosjektet skal produsere en GUI-prototype for bruksmønsteret *Reserver rom.* Dette er bruksmønsteret som du jobbet med i obligatorisk oppgave 2.

Genova er en overbygning på Rose. Genova tar utgangspunkt i entitetsklassene til systemet som du for anledningen har spesifisert i Rose, og genererer et GUI som gjør at en bruker kan registrere data for disse entitetsklassene. Genova lager automatisk kontrollobjekter (for "standardoperasjoner" som å hente, opprette, oppdatere og slette entitetsobjekter, som du senere evt. kan utvide med mer spesialisert forretningslogikk) og kantobjekter bak kulissene (dvs. bak GUIet). Det eneste som er synlig er dermed entitetsklassene og GUIet. Genova kan også lage databaseskjemaer for de av entitetsklassene som du har spesifisert som persistente. Det du som utvikler må gjøre, er følgelig å ta entitetsklassene som du har spesifisert for systemet, knytte ytterligere litt spesifikasjon til klassene, og gi disse til Genova, og så ordner Genova langt på vei resten.

I Rose må du, foruten de vanlige diagrammene som spesifiserer systemet ditt, lage et klassediagram (en såkalt navigasjonsmodell) som Genova skal lese for å skjønne hvilke GUIer som hører til hvilke deler av systemet. I ditt tilfelle skal du kun lage ett GUI for en liten del av systemet. Din navigasjonsmodell er derfor svært enkel og skal ha kun en klasse med stereotyp <<Application>> samt kun en klasse med stereotyp <<Dialog>>. (Disse stereotypene kan du sette i Class Specification-dialogboksen. Class Specification-dialogboksen har du brukt tidligere for å gi klasser attributter og metoder (operations).) Som du nok har gjettet, står <<Application>>-klassen for systemet, mens en <<Dialog>>-klasse representerer et GUI. Du kan ha flere <<Dialog>>-klasser, dersom du ønsker flere alternative GUI, eller sub-GUI for systemet ditt.

Etter noe diskusjon, har du og kunden blitt enige om at entitetsklassene for *Reserver rom* bør se ut som i figur 1. (Metodene er her utelatt av pedagogiske grunner.)

Ut fra entitetsklassene i figur 1, skal du nå definere et enkelt GUI som består av en dialog som tillater en bruker å registrere data som er relevante for en romreservasjon.



Figur 1

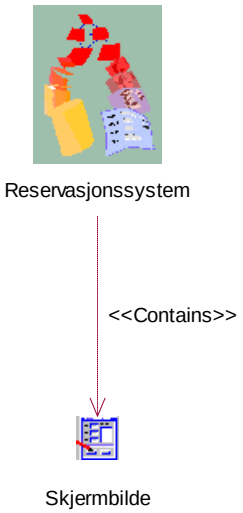
a) Domenemodell. Det første du må gjøre er å lage klassediagrammet i figur 1 i Rose (med mindre figur 1 tilfeldigvis er en del av løsningen din fra oblig 2). Hvis du har kjørt gjennom brukerveiledningen til Rose og gjort obligatorisk oppgave 2 på en ordentlig måte, burde dette gå greit. For å holde oversikten, lag først en mappe som du kan kalle "Domene" under Logical View i Rose. (Nye mapper lager du ved å høyreklikke på "Logical View" og så "New" og så "Package".) Lag så klassediagrammet ditt ifølge figur 1 (som du kan kalle f.eks. "reservasjon") i denne "Domene"-mappen. Husk å lagre dette på et passende område på hjemme-området ditt (m-disken).

NB 1: Istedenfor å lage enveis og toveis-assosiasjoner (med piler i en eller begge ender), lager du nå i stedet uspesifiserte assosiasjoner (uten pilhoder). Dette må du gjøre fordi Genova er

litt mer streng med hva disse pilene egentlig betyr enn det Rose er: I brukerveiledningen til Rose sto det at man kunne lage en to-veis assosiasjon i Rose ved å dra to enveis-assosiasjoner i hver retning. Men disse kan også tolkes som to separate enveis-assosiasjoner, og det er nettopp det Genova gjør. Merk at uspesifiserte assosiasjoner muligens ikke er synlig som ikon i toolbaren. For å legge til dette ikonet i toolbaren, går du på "View", "Toolbars", "Configure", "Class diagram – UML" og leter frem "Association" i listen på venstre side. Dobbeltklikk på den og trykk "Close", så er den lagt til i ikon-listen. Alternativt kan du trekke en enveis-assosiasjon i klassediagrammet ditt og hake av "Navigable" under både "Role A Detail" og "Role B Detail" i spesifikasjonen til assosiasjonen. Da blir den en uspesifisert assosiasjon uten pilhoder, akkurat slik du vil ha.

NB 2: I Rose sier man 0..n istedenfor 0..*. (Ting i dataverdenen er jo til syvende og sist alltid i endelige antall.)

b) Navigasjonsmodell. Så må du altså lage klassediagrammet som skal trigge Genova til å lage et GUI: Lag en ny mappe "Package" og kall den "Navigasjon". Lag så klassediagrammet som vist i figur 2 (som du igjen kan kalle f.eks. "reservasjon") i denne "Navigasjon"-mappen. I dette diagrammet må du som sagt lage en klasse med <<Application>>-stereotype (denne klassen kan du f.eks. kalle "Reservasjonsystem"), samt lage en klasse med <<Dialog>>-stereotype (denne klassen kan du f.eks. kalle "Skjermbilde"). Så skal du ha en "dependency" (striplet pil) fra "Reservasjonssytem" til "Skjermbilde" og denne dependency'en skal ha stereotype <<Contains>>. Husk å lagre. Nå har du gjort det som skal til i Rose, og du skal nå realisere dette i Genova.



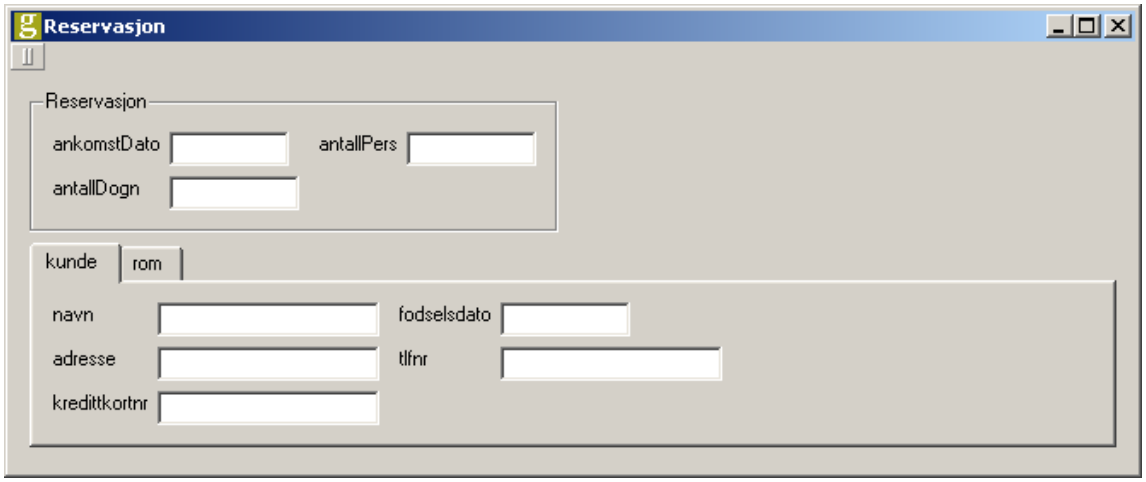
Figur 2

c) Genova. Start Genova fra Rose som angitt i Quick-starten. Lag et egnet Workspace og synkroniser med modellen i Rose på samme måte som i Quick-starten, bortsett fra at du selvfølgelig til slutt kaller Workspacet ditt for f.eks. "Reservasjon". (**NB:** Du kan sette egnete

mapper for lagring ved å gå til ”Edit -> Options” og sette ”Default Model Directory” og ”Default Workspace Directory” til ditt hjemmeområde.)

d) Objekt-seleksjon. Det er i ”object selection” (se Step 5 i Quick-starten) at felter i skjermbildet bestemmes. Det gjør du ved at du velger ut de delene (en objekt-seleksjon) av domenemodellen din som du vil ha felter for i skjermbildet. Når du nå ekspanderer ”Object selections” i Workspacet ditt (i navigasjonstreet i det venstre vinduet i Genova), vil du se at du har et (tomt) object selection som heter ”Skjermbilde_os”. Dette er laget fordi du la opp til en GUI som het ”Skjermbilde” i navigasjonsmodellen. Hadde du lagt inn flere GUI-klasser i det diagrammet, ville du ha fått opp en (tom) object selection for hver av dem. Du skal nå legge til roller i ”Skjermbilde_os”, analogt som i Quick-starten. Du skal ha en tre-struktur på GUIet med ”Reservasjon” som hovedrolle og ”Kunde”, ”Rom” og ”Hotell” som subroller. Så stå på rot-nivå og velg rollen ”Reservasjon”. Stå så på ”Reservasjon” og velg roller ”Kunde”, og ”Rom”. Stå så på ”Rom” og velg ”Hotell”.

e) Dialogmodell. Når du er ferdig med objekt-seleksjonen, genererer du dialog-strukturen (Step 6 i Quick-starten). Du skal til slutt få opp en dialogboks som ser omtrent ut som figur 3. Du skal inkludere din dialog i **<fornavn_etternavn>OBLIG3.pdf**. Dette kan du gjøre ved å ha dialogvinduet aktivt, og så gjøre Alt+Print Screen og lime inn i arbeidsdokumentet ditt. Hvis du sitter på remote desktop, kan det hende at du får med hele remote desktop-vinduet, men det er ok.



Figur 3

Oppgave 2 Database: Du skal nå lage et databaseskjema basert på klassediagrammet i figur 1. I en tidligere ukeoppgave gjorde dere mappingen fra den objektorienterte modellen til et dataorientert UML-klassediagram manuelt. Imidlertid støtter Genova denne prosessen og genererer automatisk SQL-setningene som skal til for å lage en database.

a) Domenemodell. For å få Genova til å lage et databaseskjema for deg, må du legge databaserelatert informasjon inn i domene-modellen som du laget i Rose i oppgave 1. Alle klasser som skal gi opphav til data i databasen og som derfor skal inngå i databaseskjemaet, må spesifiseres som persistente. Figur 1 er jo alle entitetsklassene dine, så da skal alle disse klassene spesifiseres som persistente. Dette gjøres enklest i Rose ved å velge "Tools -> Genova DB -> Mark All Classes Persistent". Alternativt velger man "Properties"-dialogen for hver enkelt klasse, går inn på arkivkortet "Detail" og velger "Persistent" under "Persistence". Når dette er gjort, må det velges eller lages primærnøkler for klassene. Det er hensiktsmessig å velge unike egenskaper i klassene som primærnøkler; slik som "hotellNavn" for "Hotell". Når det gjelder "Kunde", "Reservasjon" og "Rom", må det legges til attributter som er primærnøkler. Her kan en lage et nytt attributt, for eksempel kundeId: Integer og reservasjonsId: Integer. I tillegg til primærnøkler, må feltlengder også settes på de attributtene som ikke har fast lengde. Dette gjelder de elementene som er "String" og "Double" i modellen. Setting av primærnøkler og feltlengder gjøres på de aktuelle attributtene i Rose ved å åpne "Specification" og velge arkivkort "Genova DB" (som Genova-installasjonen har lagt til i Rose). Her settes "Primary Key" til "TRUE" for det attributtet i hver klasse som skal være primærnøkkel. Her settes dessuten "Length (bytes)", for eksempel 40 for å kunne lagre en tekst på 40 tegn, på attributter som er "String", og "Precision" og "Scale" på attributter som er "Double". Her angir "Precision" antall siffer foran komma, for eksempel 4, og "Scale" antall siffer bak komma, for eksempel 2.

b) Databasemapping. Resultatet av utvidelsen i modellen synkroniseres inn i Genova, og det er på tide å opprette en databasemapping. Dette gjøres ved å høyreklikke på "Database mappings" i Workspacet, velge "New Database Mapping" og gi denne et navn, for eksempel "reservasjon". Åpne databasemappingen (ved å dobbeltklikke på den) og velg "MySQL" som database. La valget "Hibernate" i Data Access Generation stå. Modellen du får opp viser alle dine persistente klasser, attributter og fremmednøkler. For klasser du ønsker automatisk genererte IDer for, som Reservasjon, kan du nå gå i database-mappingen og åpne "Properties"-dialogen for primærnøkkelattributtet, velge arkivkort "Type" og sette "Db type" til "identity". Du er nå klar til å generere databaseskjema. Men først må du sette target directory: Velg "Setup"-arkivkortet i det venstre vinduet i Genova. Gå til "Database Designer -> DBMS Schema Generation -> MySQL" og sett "Target directory" til en fornuftig mappe på ditt hjemmeområde. Trykk så på knappen "Schema" i den øverste knapperaden i Genova.

c) Databaseskjemaet. Det har nå blitt opprettet en fil i det target directory som du spesifiserte, som inneholder SQL-setninger for å opprette tabeller i henhold til klassediagrammet ditt (figur 1). Dette er med andre ord et databaseskjema. Som du ser, spesifiserer skjemaet primærnøklene som du anga, og den har faktisk generert fremmednøkler automatisk basert på assosiasjonene som du lagde i klassediagrammet. Fila heter MySQLSchema.sql, og den skal du altså levere under navnet **<fornavn_etternavn>OBLIG3.sql**

Oppgave 3 GUI: Du skal nå jobbe litt mer med GUIet.

a) Få opp et nytt preview (prototype). Oppdater først dialogmodellen din i Genova: "Dialog model->Regenerate dialog model". Trykk deretter på "Preview". Du vil se at det du har spesifisert av feltlengder i "Genova DB" i domenemodellen din i Rose nå gjenspeiler seg i feltlengdene i GUI-dialogen. Denne dialogen vil være annerledes enn den først du genererte, og du skal inkludere også denne nye i **<fornavn_etternavn>OBLIG3.pdf**.

b) Kunden er ikke helt fornøyd med GUIet. Kunden setter ikke pris på at ledetekstene til feltene tydeligvis er attributtnavnene dine fra datamodellen. Du skal derfor endre ledetekstene i skjermbildet til noe mer brukervennlig. Dette gjør du (lettest) ved å gå tilbake til domenemodellen din (i Rose) og spesifiserer GUI-ledetekstene for hvert attributt i hver klasse: For eksempel, dobbeltklikk på "Reservasjons"-klassen i Rose og velg "Attributes". Velg så et attributt (for eksempel ankomstDato) og dobbeltklikk på det. Du skal ikke endre navnet, men derimot velge "Genova UI"-fanen (som Genova-installasjonen har lagt til i Rose). Der kan du klikke på "Title" i "Model properties"-vinduet og endre på "Value"-feltet. Skriv inn et fornuftig navn, f.eks. "Ankomstdato", og trykk på "Override". Gjør slik for de andre attributtene også. Når du er ferdig med dette i Rose, går du til Genova og synkroniserer igjen.

c) I tillegg har de attributtene som du måtte legge til i klassene for å få primærnøkler til datamodellen (f.eks. kundeId og reservasjonsId) gitt opphav til felter i GUIet. Kunden er ikke sikker på at disse database-motiverte feltene trengs i GUIet (selv om de er viktige for systemet bak kulissene.) I objekt-seleksjonen din kan du gå inn og sette de aktuelle attributtene til "Excluded". Få så fram et nytt preview. Den dialogen du nå får fram, skal du også inkludere i **<fornavn_etternavn>OBLIG3.pdf**.

d) Når kunden får tenkt seg om, vil hun gjerne at reservasjonsId likevel skal være synlig på skjermbildet, slik at man kan raskt få fram en reservasjon f.eks. dersom kunden har fått oppgitt denne som en bestillingsreferanse. Du ordner derfor dette. Og du lager en mer hensiktsmessig ledetekst (f.eks. "Bestillingsref") for denne bruken av attributtet. Du bestemmer deg for å gjøre denne ledetekst-endringen i dialogmodellen (istedenfor i domenemodellen) ved å dobbeltklikke på reservasjonsID_LABEL og sette "Title" i "Properties"-boksen du får opp. Du velger å gjøre endringen i dialogmodellen, fordi du tenker deg at du senere vil lage andre skjermbilder der attributtet reservasjonsId vil spille en annen rolle og der en annen ledetekst kan være hensiktsmessig.

e) Kunden har kommet fram til at skjermbildet skal se ut som i figur 4. (Feltene er for anledningen fylt ut med et eksempel.)

Skjermbildet (dialogen) får du til som følger (du kan sjekke underveis etter hvert steg ved å gjøre "Preview", og av og til må du synkronisere på nytt osv.):

1. Sett lengden ("Genova UI -> Field Length" i attributt "Properties" i domenemodellen) til fornuftige verdier for attributtene som er av typen Integer. Det er sjelden romnumre behøver å være større enn 9999, antall personer > 99 og antall døgn > 99,

noe som gjør at du kan sette verdiene til henholdsvis 4, 2 og 2. Husk også å sette lengden på kredittkortnummer.

2. Sett layout til å være "Line3LeftFixedNoBorder" på den innerste blokka for "Reservasjon" (i dialogmodellen). Denne blokka vil hete noe sånt som "Reservasjon_DATA".

3. Flytt elementene fra de andre blokkene i dialogmodellen over i denne "Reservasjon_DATA"-blokka (på riktig sted), og slett blokker som da blir tomme. Du flytter elementer ved å dra dem til der du vil ha dem i dialogmodellen. Du sletter også de tomme "Notebook"ene.

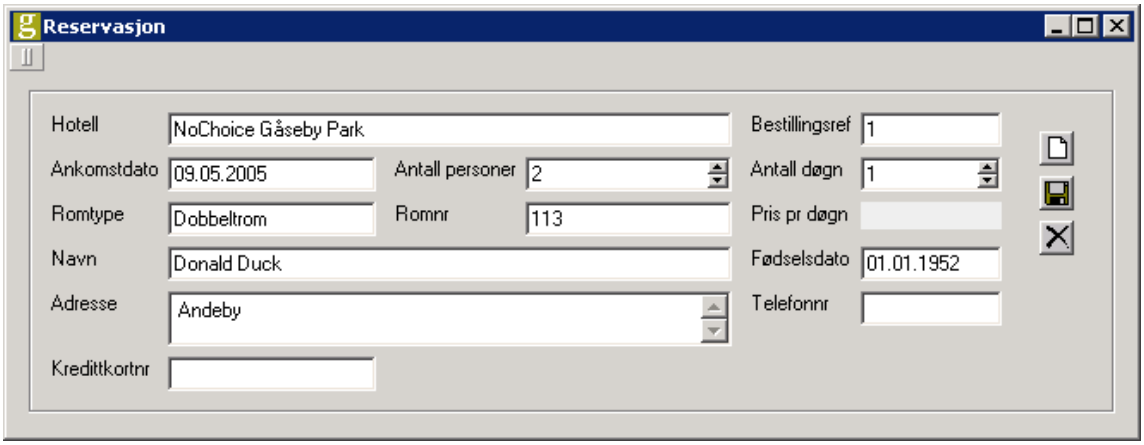
4. Sett "Column Span" til å være "2" for feltene "Hotell.hotellNavn", "Kunde.navn" og "Kunde.Adresse". (Feltene kan muligens hete noe litt annet i din modell). "Column Span" settes ved å gå inn på "Properties"-dialogen for de aktuelle elementene, velge arkivkort "Layout-overrides" og endre verdi i "Column Span" (fra 1 til 2).

5. Bytt "Style" på "Rom.dognPris" til "BlackOnLightGreyNoFrame". Dette er for å få brukeren til å forstå at feltet inneholder informasjon og ikke er et felt som kan skrives i. Sett også feltet til ikke å kunne editeres. Da kan ikke markøren settes der engang. Hvis du vil, kan du også eksperimentere deg fram til å få scroll-bokser ("stepper") for de feltene det er naturlig for (se figur 4). (Av og til skjer dette automatisk.)

6. Sett til slutt "Line3LeftStretchedNoBorder" på "Reservasjon_DATA"-blokka (den innerste blokka for reservasjon). Grunnen til du gjør dette først nå, er at du ville se hvilke elementer som bestemmer bredden på de forskjellige kolonnene.

7. Fjern tilslutt verdien i "Title" for blokka som heter "Reservasjon_BLOCK".

Fyll ut feltene med ditt eget eksempel og inkluder også denne siste dialogen i **<fornavn_etternavn>OBLIG3.pdf**.



Figur 4

Oppgave 4 Utvidelse: Kunden synes det er fint at det er mulighet for å reservere rom, men ønsker seg også en mulighet for å finne reservasjonene som finnes i systemet. Kunden ønsker seg i denne sammenheng at det finnes adresse knyttet til både kunde og hotell, slik at de kan søke på stedsinformasjon (postnummer og/eller poststed) for både hoteller og kunder. Du skal nå utvide UML-modellen og lage et nytt GUI for å søke frem reservasjoner.

a) Kunde har i dag adresse lagt inn i et enkelt felt, men siden kunden ønsker å søke på elementer innenfor adressen og også knytte adresseinformasjon til hotell, er det på tide å utvide UML-modellen med klassen Adresse. Vi trenger ikke å ta høyde for at postnumre og -steder endrer seg over tid, da vi ikke er interessert i gamle reservasjoner og reservasjoner sjelden gjøres (veldig) lang tid i forveien. Det vil si at attributtene vi trenger er gate, postnummer og poststed. Sørg for at attributtene får fornuftig type og database- og GUI-lengde, slik at vi får en fornuftig visning og lagring. Det holder at Kunde og Hotell får tilgang til en adresse, de får bruke den som er viktigst om de har flere. Attributtet adresse fra klassen Kunde kan nå slettes, siden det nå har blitt erstattet med en assosiasjon.

b) Du skal nå opprette et nytt skjermbilde for søk etter reservasjoner. Se oppgave 1b) og figur 2 for hjelp. Synkroniser utvidelsene i UML-modellen inn i workspacet ditt. Kunden ønsker å kunne søke frem reservasjoner på bakgrunn av kundeinformasjon (navn, fødselsdato eller telefonnummer), datointervall, hotellnavn og stedsinformasjon (postnummer eller poststed). Hotellkjeden vil være satt implisitt, da den som skal søke etter reservasjoner bare skal ha tilgang til reservasjoner innenfor egen hotellkjede. Det er viktig for kunden selv å kunne bestemme hvilke egenskaper det skal søkes på, så det settes ingen krav til at en spesiell kombinasjon av felter skal være fylt ut. Når det gjelder søkeresultatet, ønsker kunden at reservasjonsnummer (bestillingsref.), fradato, tildato, kunde og hotell skal vises i listen. Kopier visning av skjermbildet (”preview”) og legg det inn i **<fornavn_etternavn>OBLIG3.pdf**.

c) Generer nytt databaseskjema etter at klassen Adresse har blitt lagt til UML-modellen din. Lever skjemaet som **<fornavn_etternavn>OBLIG3_b.sql**.

SLUTT