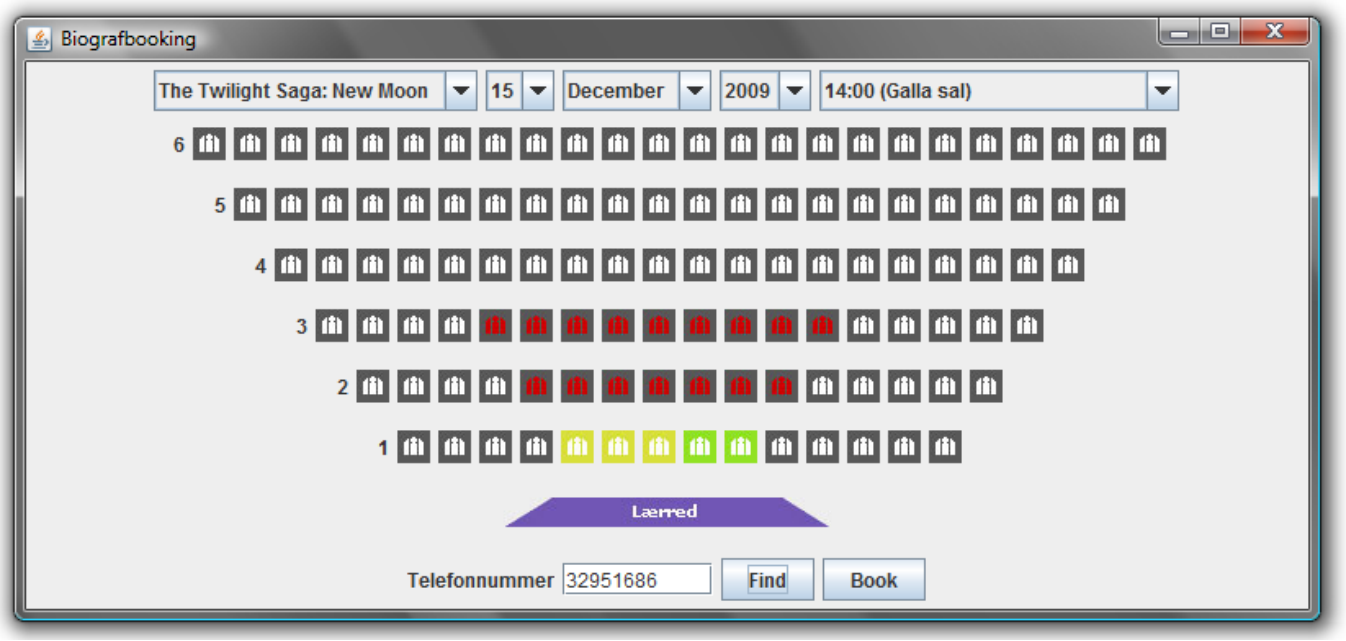
BIOGRAFBOOKING-SYSTEM

GRUNDLÆGGENDE PROGRAMMERING MED PROJEKT

AF PETER ØLSTED (050490-PTOE) OG

NICOLAI SKOVVART (231288-NSBK)



VEJLEDER: RASMUS EJLERS MØGELBERG

INDHOLDSFORTEGNELSE

FORORD OG INDLEDNING

Denne rapport er udarbejdet i december 2009 i forbindelse med et to-ugersprojekt på IT-Universitetet i Købenavn under vejledning af Rasmus Møgelberg.
I projektet udviklede vi et Java-program til booking af billetter i en biograf til brug af en ekspedient. Programmet kan reservere sæder efter telefonnummer og vise filmforestillinger baseret på hvilken film der vises, salen den vises i, dag, måned, år & klokkeslæt med dertilhørende reservationer. Programmet gemmer disse data i en database og henter relevant data efter behov.

PROBLEMSTILLING OG BAGGRUND

Vi blev stillet opgaven at programmere et Java-program med en grafisk brugergrænseflade der var defineret ud fra en opgavetekst.

Systemet skulle bruges til at håndtere bestilling af billetter af en ekspedient der sad i billetlugen i en biograf, men skulle ikke håndtere salg. Systemet skulle kun blive brugt af ekspedienten og behøvede derfor ikke at tage hensyn til dataopdateringer fra andre.

Systemet skulle understøtte ekspedientens arbejdsopgaver, som f. eks. indebærer:

- Bestilling af billetter
    - til en bestemt forestilling (film og tidspunkt)
    - bestilling til film (men hvor tidspunktet er underordnet)
    - bestilling til tidspunkter (men hvor filmen er underordnet)
    - bestilling til en stor mængde reservationer (men hvor film og tidspunkt er underordnet)
- Afbestilling af reservationer
- Ændringer til reservationer
    - fjerne/tilføje sæder
    - rokering af reservationer i biografen
    - til en tidligere/senere forestilling

Vi besluttede at bruge skabelonen i projektbeskrivelsen som basis for den grafiske brugergrænseflade, men valgte dog at lave nogle ændringer til menuerne og deres placering.

Vi besluttede for at benytte en MySQL database til opbevaring af alt vores data – biografsalenes opbygning, film, forestillinger og reservationer. Vi besluttede også at en reservation kun gjaldt for et enkelt sæde, og at flere sæder derfor ville kræve flere rækker i databasen.

Vi valgte at opdele programmets logik. Vi lavede en DBHandler der skulle håndtere forbindelsen til databasen samt at køre alle de queries vores program skulle understøtte, et CinemaGUI der skulle kunne fremvise en biografsal grafisk og en WindowHandler der skulle holde vores knapper og dropdownmenuer, samt CinemaGUI'et.

Senere tilføjede vi en klasse til håndtering af tid (gemt som UNIX timestamps i databasen) samt flere mindre klasser til data-opbevaring.

Ved bestilling af billetter skulle ekspedienten vælge en film og en dato for at se på hvilke tidspunkter der var forestillinger. Efter at have valgt en forestilling skulle programmet tegne biografsalen med de bookede sæder markeret. Hvis der ikke var forestillinger til filmen på denne dato, skulle dropdown-menuen fortælle dette og ikke tegne nogen biografsal.

Ved afbestilling af billetter skulle ekspedienten vælge forestillingen der skulle afbestilles fra og søge på det telefonnummer der var booket på. Sæderne burde herefter markeres, og man kunne så klikke på sæderne hvilket ville fjerne bestillingen. Hvis man ikke kendte telefonnummeret kunne man ikke afbestille sæder.

Ved ændring af billetter skulle ekspedienten vælge forestillingen, søge på telefonnummeret og derefter vælge flere/færre sæder og trykke book.

PROBLEMANALYSE

BRUGERGRÆNSEFLADEN

VINDUER

Omkring brugergrænsefladen valgte vi at samle alt i ét vindue noget som giver både fordele og ulemper. Fordele: ét-vindue giver en klart simplere brugergrænseflade og giver mulighed bedre muligheder for at bruge tiden på ét vindue i stedet for at sprede tiden over flere. Dette design giver har også den oplagte mulighed at åbne flere instanser af samme vindue. Hvis vinduet også holder en relativ lille størrelse kan der åbnes mange instanser af samme vindue på en enkelt skærm.

Ulemper: det kan gøre oversigten dårligere. Hvis man ønsker at finde en oversigt over en persons reservationer eller en længere liste over film er det ikke længere muligt. Generelt bliver oversigter ikke længere muligt medmindre man laver et meget stort vindue der så ender med at understøtte mange arbejdsopgaver i samme vindue. Dette har så problemet at det hurtigt bliver uoverskueligt hvis der er 15-20 knapper/dropdown lister i samme vindue.

Den primære grund til vi har valgt et ét-vindue-design er også for at gøre det så fokuseret som muligt på hovedopgaven, at booke sæder. I et professionelt kan det bestemt være praktisk at have en oversigt, men det er ikke nødvendigt i et lærings og et ikke-kommercielt program.

BIOGRAFSALEN

Omkring tegningen af biografsalen har vi diskuteret to mulige løsninger til at tegne det. Der er flere løsningsmuligheder, men dette er de to vi har kigget på.

Den første mulighed er at bruge et canvas og bruge paint() metoden til at tegne en sal efter et imaginært gitter og så beregne hvor i gitteret brugeren klikker.

Dette har den store fordel at man har fuldstændighed frihed til at tegne sæderne præcis hvor man ønsker det. Hvis du ønsker at lave en biografsal hvor sæderne former et stort X er det muligt. Det kræver så også at databasen har informationer om hvordan sæderne helt præcis er placeret og ikke bare specificerer for hver række hvor mange pladser der er.

Et problem ved dette er dog det kan give et mere kompliceret system da hvert element ikke i samme grad kan opdatere sig selv og derved vil man få en kode med en større kobling end nødvendigt.

Den anden mulighed er at tilføje billeder i rækker af forskellige længder og lave et ”linjeskift” når en ny række skal tegnes. Her tjekker hvert billede om der bliver trykkes på det, og det har til opgave at opdatere sig selv med de rette billeder.

Her får alle billederne en mouseListener som giver os mulighed for at billederne kan opdatere sig selv.

Dog når man tegner billederne har vi ikke samme fleksibilitet da det kræver vi bruger Javas layout managers. Flowlayout managers giver dog ikke mulighed for at lave noget efter et fast mønster, så hvis vi ønskede fleksible sale med denne løsning skal alle rækkerne have et enten lige eller ulige antal sæder. Hvis ikke ville sæderne være forskudt i forhold til hinanden.

Vi endte med løsning nummer to da vi ønskede en fleksibel sal dog uden at gøre det unødvendigt kompliceret. Problemerne med sæderne har vi arbejdet udenom ved at gøre alle rækkerne i salene have et enten lige eller ulige antal sæder.

VALG AF DATO

Omkring datoen havde vi to muligheder, enten lave en enkelt dropdown med en liste over de næste 14-20 dage eller frit kunne vælge datoen via tre forskellige lister. Valget her var rimelig frit, da vores kode endte med at kunne understøtte begge muligheder uden noglenævneværdige ændringer i koden.

Forskellen mellem dem ligger mest i kompromiet mellem hastighed og fleksibilitet. En liste over de næste 20 dage er meget nemt at håndtere for brugeren. Det er meget nemt at finde en dato f.eks. en uge frem i tiden, men til gengæld kan du ikke vælge en dato 50 dage frem i tiden da det ville give en meget lang og uoverskuelig liste. Med tre lister kan man frit vælge datoen, både før og efter den nuværende dato. Dette giver mulighed for at reservere en billet til premieren på den næste blockbuster. Vi ved dog ikke hvor normalt det er at reservere en billet 50 dage frem i tiden, så det kan være et usandsynligt eksempel.

Vi valgte designet med dag/måned/år menuer af én grund, den større fleksibilitet som vi syntes er vigtigere end den smule ekstra nemhed det andet system ville give.

Til behandling a tid benytter vi biblioteket Joda time, da den var stærkt anbefalet på internettet. I bagsyn kunne vi dog sikkert have genskabt den samme funktionalitet ved brug af Javas indbyggede Java.util.Calendar.

DATABASEVALG

For at gemme dataen programmet genererer, ville vi gemme det i en database. Valget af database stod mellem Microsoft Access, MySQL og Oracle. På grund af tidligere erfaring med databaser valgte vi MySQL da vi begge havde erfaring med denne. MySQL og Oracle havde også den fordel at IT-Universitetet tilbyder databaserne på sine servere, så vi kunne oprette forbindelse til databasen fra alle computere med internetforbindelse.
En sidste fordel er at MySQL er meget udbredt, så vores database ville hurtigt kunne forbindes med andre systemer.

DATABASEDESIGN

For at lave et biografsystem er der overordnet set brug for en måde hvorpå man kan gemme reserveringer til en bestemt forestilling på.
Vi har valgt at gemme de film der vises, hvilke sale der findes og hvilke forestillinger der vises i hver sin tabel for at undgå dataduplikering samt for nemt at kunne tilføje nye af disse.
En forestilling i vores design er en samling af tre informationer, hvilken film der vises, hvilken sal den vises i og hvornår visningen foregår.

Der er mindst to måder man kunne opbygge en biografsal på. Den første tilgang er at gemme direkte i tabellen hvor stor en sal er via to variabler, antal rækker og antal sæder per række. Når salen skal tegnes tegner man antal rækker gange antal sæder per række og får herved en rektangulær sal.
En anden tilgang er at have en tabel med rækker der tilhører en sal, og en tabel med sæder der tilhører en række. Denne tilgang giver et mere fleksibelt system og lader hver biografsal være fuldstændig unik. Alle biografer kan have et variabelt antal rækker, og hver række kan have et variabel antal sæder per række.

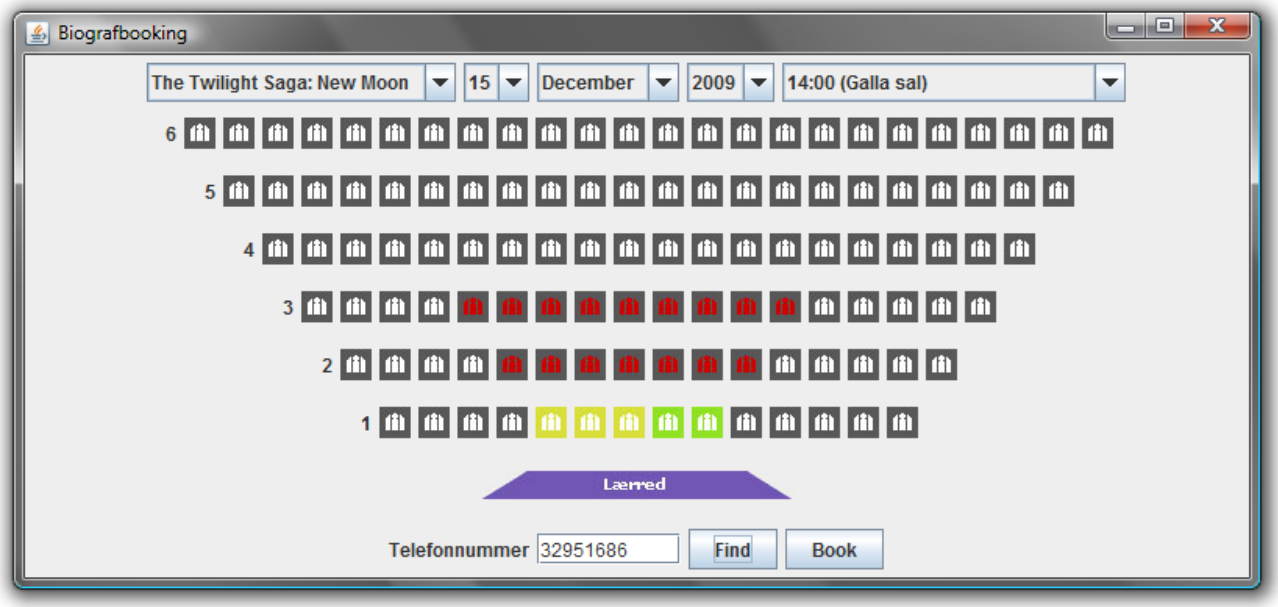
Vi valgte at bruge den anden tilgang på grund af den øgede fleksibilitet.

En forestilling har i databasen en sal, en film og et tidspunkt. For et undgå at filmen og salen bliver duplikeret i databasen peger disse værdier på en unik identifikations-nøgle for den respektive film og sal i stedet for navnet på salen og på filmen.

Reservationer gemmes i en tabel for sig selv og der bliver tilføjet en række for hver sæde der bliver reserveret. Reservationer peger på et sæde (der tilhører en bestemt biografsal) og på en bestemt forestilling. For hver reservation gemmer vi også telefonnummeret. Dette kan give noget duplikeret data, men forsimpler datamodellen en smule og reducerer antallet af queries der skal køre.

Databasedesignet understøtter alle arbejdsopgaverne beskrevet i projektbeskrivelsen, men kunne idéelt udvides til at håndtere salg , for eksempel med priser baseret på film, forestilling og/eller pladsering af sæderne.

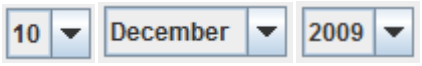
BRUGERVEJLEDNING TIL BIOBOOKING



VÆLG EN FORESTILLING

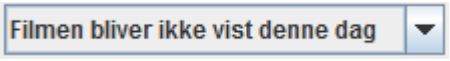
Vælg en film (A) og en dato (B) (i vilkårlig rækkefølge) fra deres respektive dropdown-menuer:

(A) 

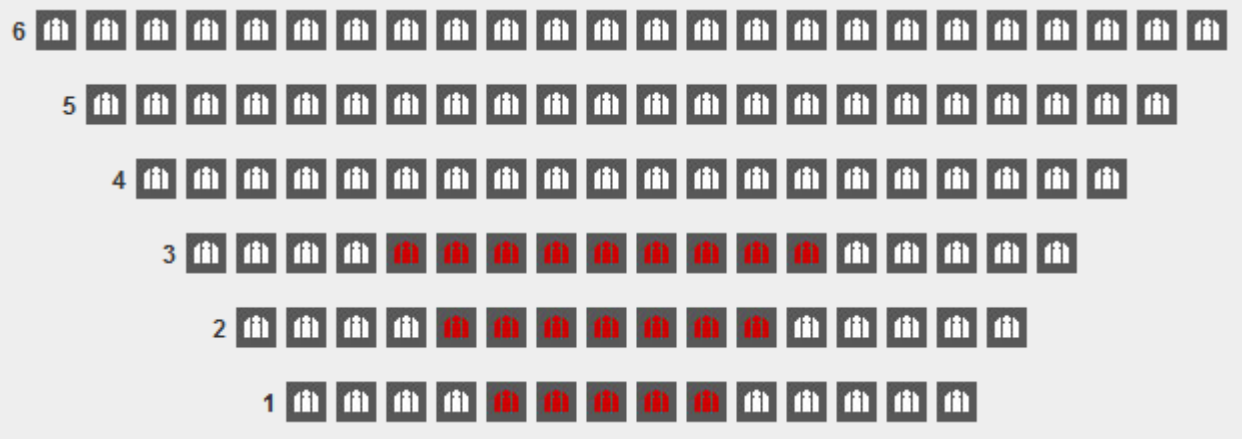
(B) 

Derefter opdaterer forestillingslisten ude til højre (C) fakta om hvor du kan se hvor man se hvilken sal og tidspunkt forestillingen bliver vist på (hvis filmen bliver vist på denne dag, ellers vises en meddelelse der forklarer at filmen ikke bliver vist denne dag (D)).

(C) 

(D) 

Efter at have valgt en forestilling får man nu et billede af biografsalen til denne forestilling.



Frie sæder bruger  ikonet , og bookede sæder bruger  ikonet.

RESERVER SÆDER TIL FORESTILLING

Vælg den forestilling kunden vil reservere sæder til. Klik på de frie sæde(r) du vil reservere. Valgte sæder ændrer billede fra  til  . Indtast kundens telefon nummer i telefonnummer tekstfeltet (D). Telefonnummeret bruges til afhentning af billetter, til at finde reserverede sæder og kan (hvis funktionaliteten tilføjes) bruges til at føre statistik over besøgende.

(D) 

Efter at have indtastet telefonnummeret kan du nu trykke på ”Book” hvorefter sæderne bliver booket (E).

(E) 

ÆNDRE RESERVATION

Vælg den forestilling hvor du skal ændre reservationen og indtast derefter kundens telefonnummer. Tryk på ”find” knappen, og hvis der findes nogle reservationer med kundens telefonnummer bliver de nu markeret med  ikonet. For at slette reservationen, tryk på ikonet og reservationen er slettet med det samme. For at tilføje flere sæder (eventuelt ved siden af), tryk på de frie sæder og tryk ”Book”.

TEKNISK BESKRIVELSE AF PROGRAMMET

INTERNE DATASTRUKTURER – DATABASE

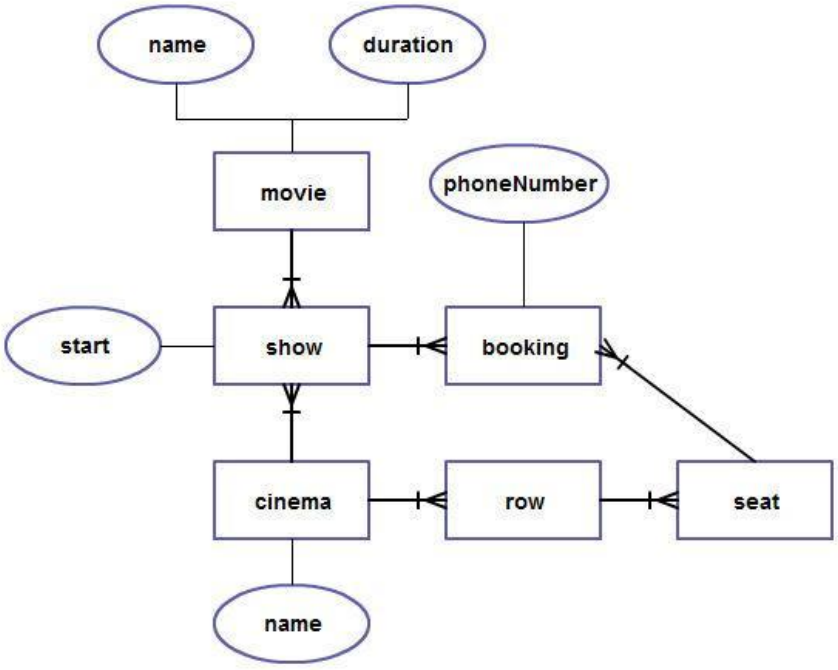
Biografsalene (”cinemas”) er opbygget af et variabelt antal rækker, og hver række har et variabelt antal sæder (”seats”). Navnet på biografsalen kan ses ved siden af start-tidspunktet i dropdown-menuen der viser forestillinger.

Film har et navn (”name”) og en varighed (”duration”) i minutter. Varigheden bliver ikke brugt i programmet pt., men vil kunne bruges til hurtigt at generere en mængde forestillinger på en dag og til at sørge for at man ikke planlægger forestillinger når den foregående film stadig vises. Filmtitlen vises i dropdownmenuen for film og sorteres i alfabetisk rækkefølge.

Forestillinger (”shows”) har et starttidspunkt (UNIX timestamp i sekunder) og peger på én film og én biograf. De vises i en dropdown-menu baseret på den valgte film og den valgte dato.

Bestillinger (”bookings”) peger på én forestilling og ét sæde, og disse data kædes sammen med et telefonnummer (som bestillinger kan hentes på, og som kan bruges hvis bestillingen skal opdateres senere). Bookede sæder bliver vist i programmet med  ikonet eller med  hvis man har søgt på et telefonnummer.

Data-forholdet er illustreret i ER-modellen nedenfor.



INTERNE DATASTRUKTURER – KLASSERNE

Programmet primære opbygning består af tre klasser med hver deres distinkte formål.

DBHandler er vores forbindelse til databasen. Alle metoder indsætter data eller der henter data fra databasen går igennem denne klasse og hver gang en anden klasse har behov for at hente data fra databasen blev der oprettet en metode til det. F.eks. henter metoden getShows() alle forestillingers der går en specifik dag og til en bestemt film.
For at sikre at der altid kun er én forbindelse til databasen har vi lavet klassen efter designmønsteret Singleton ved at lave constructoren private og have en statisk metode der henter en reference til en instance af DBHandler.
DBHandleren sikrer sig fra at miste forbindelse hvis forbindelsen er timet ud. I hver metode tjekker vi om forbindelsen er lukket. Hvis forbindelsen er lukket og det ikke lykkes at oprette en forbindelse fanger vi også en SQLException exception og printer fejlen, så man kan se at det ikke lykkes at genoprette forbindelsen.

WindowHandler har til ansvar at oprette vinduet, tilføje elementer og deres eventhandlers (hvis de ikke gør det internt).

Programmet har fem droplister - de fire af dem opdaterer den femte liste som er listen over forestillinger der viser en bestemt film for kun for den valgte dag. Der findes listen over alle film biografen viser, en liste over dage, måneder & år og til sidst den føromtalte liste med forestillinger.
Alle dropdown-menuer reagerer kun på egne ændringer via en ItemListener der mere specifikt kræver at man implementerer itemStateChanged. itemStateChanged bliver kaldt hver gang der markeres et nyt element på listen, både når en bruger trykker på listen eller systemet fjerner alle elementer på listen og også når de bliver tilføjet igen.
Resten af listerne bruger også itemStateChanged og alle kalder metoden updateShowBox() der har til ansvar at opdatere forestillingslisten og fjerne oversigten over den aktive sal hvis der ikke er nogen forestillinger med de valgte data.
Nederst i vinduet er der (som bekendt) et felt og to knapper. Feltet er lavet til at kun acceptere tal da det er hvor brugeren indtaster telefonnummeret der skal reserveres under eller som er blevet reserveret tidligere. Dette er lavet ved at lave en klasse der udvider JTextField (Java-bibliotekets standart tekstfelt klasse).

For at sikre der kun bliver indtastet tal af brugeren, har vi lavet keyCheck metoden der kun accepterer tal (og backspace og delete).

```
private boolean keyCheck(char c){
    return ((c == KeyEvent.VK_BACK_SPACE) || (c == KeyEvent.VK_DELETE) || (Character.isDigit(c)));
}
```

Hvis funktionen ikke ikke returnerer true (altså, hvis man har indtastet noget som ikke er et tal, backspace eller delete) consumer vi tegnet så det ikke bliver tilføjet til tekstfeltet.

```
if(!keyCheck(e.getKeyChar())){
    e.consume() ;
}
```

Hver gang en tast bliver trykket på eller sluppet tjekker vi også længden af indholdet i feltet. Hvis det overskrider længden af telefonnumre (som er defineret som et final int feldt) vil indtastningen også blive consumet i WindowHandler.
"Find"-knappen fungerer ved at konvertere indtastningen i tekstfeltet (en tekststreng) til et tal hvor vi så gentegner en biografsalen og passer telefonummeret til dets constructor hvor den så internt tjekker sæderne efter nummeret. "Book"-knappen henter en liste over de nye sæder der skal reserveres og passer sædernes ID'er til DBHandler som derefter reserverer sæderne.

WindowHandler holder også styr på antallet af vinduer der åbne da systemet understøtter at der åbnes flere vinduer der mangler bare en knap til at åbne et nyt vindue. *numberOfOpenWindows* er en statisk variabel i WindowHandler der henholdsvis bliver talt op/ned når vinduer bliver åbnet/lukket. Når det sidste vindue bliver kalder den DBHandlerens luk metode der afbryder forbindelsen til databasen, samt bliver System.exit() kaldt, så Javas virtuelle maskine lukker programmet.

CinemaGUI er klassen der tegner selve biografsalen som sæderne og lærredet. Den henter ikonerne, der repræsenterer biografsæderne, fra billedfiler vi selv har lavet. CinemaGUI arver fra JPanel så vi kan tegne det som et ethvert andet JPanel og integrere det nemmere med WindowHandler.
Hver gang brugeren vælger en ny forestilling bliver en ny CinemaGUI oprettet, og ID'et for det show der skal vises samt det telefonnummer brugeren har tastet ind parset til objektets constructor.

SeatStates er en enum klasse der bruges til holde styr på hvilke sæders reservation der bliver ændret når brugeren klikker på dem. Når der klikkes på et sæde bruges et HashMap (Nøgle: sæde id, Værdi: status> til at gemme de forskellige sæders opdaterede status. Hvis et sæde blive afreserveret bliver statusen for det sæde henholdsvis tilføjet til HashMap'et og herefter ændret værdien til REMOVED_RESERVATION_SEAT, altså at det ikke længere reserveret, men har været.
SeatStates blev på et tidspunkt brugt til at styre hvilket billede sæderne skulle bruge, men pga. et problem med at afmærkere sæderne igen blev denne kode fjernet.

To steder bruger vi selv exceptions og ikke bare fanger exceptions som JDBC koden kaster. I dropdown-menuen over forestillinger prøve vi at caste et *Object* som vi henter fra listen. Hvis det ikke kan castes er det en String der er blevet tilføjet til listen (og den indeholder en besked) og intet skal gøres ud over exceptionen skal fanges via en *ClassCastException.*

MovieHolder og ShowHolder er små klasser brugt til nemmere at have adgang til information mellem klasserne og for at printe information i dropdown-menuerne. Begge klasser overskriver toString() metoden nedarvet fra *Object*[1] og toString() bruger vi til at kunne tilføje disse klasser til deres respektive dropdown-menuer. Dropdown-menuerne kalder så internt vores objekters toString() metode.

INTERNE ALGORITMER

Java timestamps (millisekunder) bliver konverteret til UNIX tid (sekunder) ved brug af division med 1000. Vi gør dette da UNIX-time (sekunder fra UNIX-epoken klokken 00:00:00 1 januar 1970) er en mere udbredt standard og man derfor vil kunne udnytte dataen i andre programmer (og programmeringssprog), og da vi ikke har behov for millisekund præcision. UNIX timestamps fylder derudover også mindre i databasen.

BRUGERGRÆNSEFLADEN

Brugergrænsefladen er opbygget af WindowHandler klassen og CinemaGUI klassen.

LAYOUT MANAGERS OG KOMPONENTER

WindowHandler klassen laver en JFrame med BorderLayout.

I nord-positionen tilføjes et FlowLayout med JComboBox'es til film, dag, måned, år og forestillinger.

I center-positionen tilføjes et CinemaGUI. CinemaGUI benytter et GridLayout(0, 1). CinemaGUI tilføjer rækker af JLabels med FlowLayout og til disse rækker tilføjes en JLabel med rækkenummeret efterfulgt af JLabels med billed-ikonerne. Efter rækkerne er blevet lavet, bliver en JLabel med billedet af biografsalens lærred tilføjet.

[1] Som alle klasser i Java nedarver fra

I syd-positionen tilføjes et FlowLayout med en JLabel med ordet ”Telefonnummer”, et NumericTextField (en udvidelse af JTextField der kun accepterer numre), samt to JButton’s (”Find” og ”Book”).

LYTTERE

JComboBox’ene bruger ItemListeners der reagerer på itemStateChanged.

Sæderne bruger MouseListeners der reagerer på mouseClicked.

NumericTextfield’et benytter en KeyListener der reagerer på keyTyped og keyReleased for at sørge for at man ikke kan booke/søge på forkert-formatterede telefonnumre.

JButton'sene benytter ActionListeners der reagerer på actionPerformed.

AFPRØVNING

TEST AF TELEFONNUMMER FELTET

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
Tal mellem 0-9	Det indtastede tal viser sig i feltet.	Det indtastede tal viser sig i feltet.	Ingen
Alle bogstaver	Intet sker	Intet sker	Ingen
Kopieret tekst ind	Teksten bliver kopieret ind (ønsket funktion: intet sker)	Den kopierede tekst står i feltet	Selvom feltet ikke accepteret indtastning af bogstaver, kan man kopiere alt hvad man ønsker ind i feltet.
Der står 7 tal eller under i telenummerfeltet	Find og ”Book”-knappen kan ikke klikkes på	Find og ”Book”-knappen kan ikke klikkes på	Ingen
Der står 8 tal i telefonnummerfeltet	Find og ”Book”-knappen kan klikkes på	Find og ”Book”-knappen kan klikkes på	Ingen
Der er indtastet et tal hvorefter at bogstav er kopieret ind i stedet for et eller flere af tallene	Find og ”Book”-knappen kan ikke klikkes på	Find og ”Book”-knappen kan klikkes på	Der bliver ikke tjekket for indholdet af tekstfeltet, men kun om hvert indtastet tegn er

			gyldigt. Det betyder man kan omgå sikringen. Når der klikkes på find eller ”Book” vil programmet kaste en NumberFormatException exception da der internt er forventet et tal er kommet som input, ikke en streng.

TEST AF FINDKNAPPEN

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
Et gyldigt telefonnummer og en forestilling uden nogle reserveringer til dette telefonnummer.	Intet sker	Intet sker	Ingen
Et gyldigt telefonnummer og en forestilling med en reservering til dette nummer	De rette sæder bliver markeret med det rette gule ikon.	De rette sæder bliver markeret med det rette gule ikon.	Ingen
Et ugyldigt telefonnummer (bogstaver i det)	Intet sker	Programmet kaster en NumberFormatException exception.	Da programmet ikke kan håndtere en tekststreng som input vil komme en fejl, men hvis teksten bliver slettet og et gyldigt telefonnummer skrives fungerer programmet stadig.

TEST AF ”BOOK”-KNAPPEN

Antagelser: Et gyldigt telefonnummer er indtastet, ellers vil samme fejl ske som ved Find-knappen

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
Der klikkes på et sæde der er reserveret (rødt ikon)	Intet sker	Intet sker	Ingen
Der klikkes på book knappen uden nogen sæder er valgt	Intet sker	Intet sker	Ingen
Der er klikkes på et eller flere tomme sæder og der klikkes på ”Book”-knappen	Sæderne bliver booket i databasen og der sæderne skifter ikon til booked (rødt ikon)	Sæderne bliver booket i databasen og sæderne skifter ikon til booked (rødt ikon)	Ingen

TEST AF SÆDERNE

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
Der er fundet en reservation (gult ikon) og der klikkes på sædet	Reservationen bliver slettet i databasen og ikonet skifter til sædet er frit (gråt ikon)	Reservationen bliver slettet i databasen og ikonet skifter til sædet er frit (gråt ikon)	Ingen

TEST AF VALG AF FILM

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
En film bliver valgt og der er en forestilling med den viste film på den valgte dato	Visningen af den korrekte forestilling hvor de rette sæder er markeret som reserveret	Visningen af den korrekte forestilling hvor de rette sæder er markeret som reserveret	Ingen
En film bliver valgt, men der findes ingen	Listen over forestillinger skriver ”Filmen bliver	Listen over forestillinger skriver ”Filmen bliver	Ingen

forestilling med den valgte film på den valgte dato	ikke vist denne dag"	ikke vist denne dag"	

TEST AF VALG AF FORESTILLING

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
En forestilling bliver valgt af brugeren	Visningen af den korrekte forestilling hvor de rette sæder er markeret som reserveret	Visningen af den korrekte forestilling hvor de rette sæder er markeret som reserveret	Ingen

TEST AF VALG AF DATO

Inddata	**Forventet uddata**	**Faktisk uddata**	**Uoverensstemmelser**
Den valgte dato har en forestilling med den valgte film	Visningen af den korrekte forestilling hvor de rette sæder er markeret som reserveret	Visningen af den korrekte forestilling hvor de rette sæder er markeret som reserveret	Ingen
Den valgte dato har ikke en forestilling med den valgte film	Listen over forestillinger skriver "Filmen bliver ikke vist denne dag"	Listen over forestillinger skriver "Filmen bliver ikke vist denne dag"	Ingen

Systemet virker som ønsket så længe at databasen ikke bliver ændret uden om systemet, og så længe man ikke copy-paster tekststrenge der ikke er tal (på grund af at systemet forventer at telefonnummeret kan konverteres til en int). I de tilfælde fungerer systemet ikke efter hensigten.

Det er også muligt at copypaste numre der er længere end 8 cifre (så længe at tallet er inden for en ints grænser), og på trods af at systemet stadigt virker, så er det ikke hensigten.

Vores system tillader i store træk ikke forkerte input fra den grafiske brugerflade, og vores tests dækker de fleste brugssituationer. For at få systemet til at fejle skal man manuelt rode rundt i databasen eller udnytte at NumericTextField'et ikke håndterer copy-paste optimalt.

KENDTE FEJL

Alle tekststrenge er skrevet direkte ind i programkoden, så der er umuligt at intallere programmet i andre lander, da de ikke taler dansk. Hvis dette skulle ændres ville vi oprette en klasse der indeholdt alle nødvendige tekststrenge og f. eks. gemme dem i databasen eller i en tekstfil. Det ville sørge for at sproget hurtigt kunne ændres uden at kræve adgang til kildekoden.

Nogle gange, når man holder musen over de forestillinger der findes den markerede dag, er programmet ekstremt langsomt til at flytte musen. Programmet opfører sig som om den laver meget arbejde hver gang man bare flytter musen. Hvorfor dette sker ved vi ikke.

Når man ændrer den måned der er aktiv i programmet, opdaterer den dropdown-menuen med forestillingerne denne dag et uhensigtsmæssigt antal gange. Optimalt ville det kun ske ét gang, men pga. at forestillingslisten bliver opdateret hver gang datoen ændrer sig, bliver den opdateret seks gange for hver gang en anden måned bliver valgt.

Hvis man har flere vinduer åbne i samme sal og der bliver lavet en ændring i reservationerne i det ene vindue, kan man ikke se ændringen i det andet før programmet opdaterer billedet af salen. Dette betyder at der kan reserveres flere gange på de samme sæder. Hvis begge vinduer markerer samme sæder og booker dem, vil der blive gemt reserveringer for flere numre på samme sæder. Hvis en af reservationerne på det dobbeltbookede sæde slettes vil alle reservationer til det sæde bliver fjernet.

Når vores databaseforbindelse kaster en exception, bliver de fanget, men der bliver ikke gjort noget ved det. Brugeren får intet at vide om der er gået noget galt medmindre Java-konsollen er slået til i kontrolpanelet, noget som er meget usandsynligt på en computer der ikke bliver brugt til udvikling i Java. Optimalt ville programmet prøve at køre den metode der gik galt igen (hvis passende) og prøve at fikse grunden til at fejlen opstod og til sidst informere brugeren at der internt i programmet gik noget galt og bede brugeren om at prøve igen eller kontakte supporten.

Vinduestørrelsen matcher ikke antallet af sæder hvis der er irregulære former.

For at markere at sæde skal musen være helt stille og må ikke bevæge sig mellem knappen bliver trykket på og sluppet igen. En bedre løsning ville være at markere et sæde så længe musetasten blev trykket ned og sluppet over samme sæde. Dette er dog en konsekvens af at bruge MouseAdapter fra Javas event-bibliotek.

KONKLUSION

Programmet fungerer og kan bruges til alle de beskrevne arbejdsopgaver – dog kan det blive set som en mangel, at man skal kende film, dato, tidspunkt og telefonnummer for at kunne ændre/slette reservationer.

Vi mener selv at programmet er meget intuitivt. En hurtig usabilitytest på en medstuderende der ikke havde set vores program før bekræftede til dels dette (vi erkender dog at en power-user , der selv har arbejdet med et biografbooking-system, ikke er den idéelle forsøgsperson).

Det ville have været idéelt at have haft et separat søgevindue der kunne bruges til mere avanceret brug (f. eks. at finde ud af hvornår man kan se film ”x” med 10 ledige pladser) og til at finde reservationer hvor kunden ikke kan huske til hvilken tid han har bestilt.

Programmet har nogle problemer med hastighed, idet at show-boksen bliver opdateret uhensigtsmæssigt mange gange. Vi ville sikkert også kunne gøre programmet hurtigere, hvis vi nøjedes med at hente salenes opbygning en enkelt gang og benytte dataen fra hukommelsen, i stedet for at spørge databasen om salens opbygning hver gang vi ændrer show.

Selve programmeringen kunne med god effekt have benyttet arv bedre, for eksempel ved sæderne i biografen. Det ville gøre det lettere at udvide (hvis vi nu f. eks. ville tilføje funktionalitet for ”betalte reservationer”), og ville sikkert også mindske kodeduplikation en del. I det hele taget kunne alle events for sæder nok sagtens klares af sædet selv baseret på dets subklasse, og ikoner og anden information kunne gemmes direkte i subtypen af ”sæde”.

WindowHandler og DBHandler er relativt store klasser i forhold til deres funktionalitet, og kunne sikkert nemt have blevet optimeret. De fungerer dog.

Programmet mangler lokalisering – alle tekststrenge er på dansk, og systemet accepterer kun telefonnumre med 8 cifre (begge ting er relativt simple at opdatere, men burder idéelt være uden for systemet, f. eks. i databasen eller i nogle konfigurationsfiler).

Programmet benytter også en hardcoded forbindelse – dette vil være relativt let at opdatere, og fungerer fint til at demonstrere projektet, men er ellers ikke idéelt.

Generelt er vi tilfredse med vores program, men vi er overbevist om at vi kunne have klaret det bedre med mere tid stillet til rådighed og med en klarere programstruktur til at begynde med.

REFLEKSION OVER PROCES

Vi startede programmeringen fredag den 27 uden at have gjort nogle større tanker om programmet, andet end vi havde tænkt os at benytte MySQL da vi havde tidligere erfaring med denne database.

Vi startede uden nogen gruppekonstitution eller nogen overordnet tidsplan. Vi besluttede os for at møde klokken 12 og fokuserede i den første uge fuldstændigt på programmet.

Efter nogle dagle ændrede vi starttidspunktet til klokken 11 og vi havde opnået konsensus om at arbejde til klokken 16.

På trods af at vi ikke har haft noget behov for sanktioner, da vi begge har haft nogenlunde de samme ambiotioner, arbejdstempo, erfaring og "seriøsitet" omkring projektet, så ville det sikkert have været en god idé at have haft en fast grupekonstitution.

Vi brugte heller ikke nok tid på at overveje programmets struktur – på trods af at vi har haft glæde af, og har været konsistente med, brugen af DBHandler, CinemaGUI og WindowHandler, så er der stadig meget tid der kunne have været sparet hvis vi havde overvejet hvordan vi ville opbevare og håndtere meget af dataen (som f. eks. film, show, valgte sæder osv.).

På grund af vores manglende overordnede tidsplan blev vi nødt til at stoppe med programmeringen af programmet (som vi ellers begge var interesserede i at forbedre) og begynde på at skrive rapporten. Vi begyndte at opdele afsnit begyndte ellers at skrive, hvorefter vi læste korrektur på og godkendte hinandens arbejde (og arbejdede videre på detaljer som kunne forbedres).

Når vi var uenige havde vi heller ikke nogen protokol at følge, men vi har ikke haft nogle større problemer med at finde en løsning vi begge var tilfredse med. En gruppekonstitution ville kunne have haft en protokol der skulle følges i tilfælde af at vi ikke kunne have opnået enighed.

Havde projektet været større (gruppestørrelse eller tidsperiode) ville det sikkert have skabt væsentlige flere problemer end vi heldigvis løb ind i.

Vi havde begge to erfaring med programmering, men har trods alt lært en del om Java-specifik programmering og om at arbejde på programmerings-opgaver i grupper.

BILAG

KONDENSERET ARBEJDSDAGBOG

Dato	Noter
27/10	Generel programstruktur planlagt: WindowHandler til det generelle GUI, CinemaGUI til biografsaldisplayet og DBHandler til håndtering af databasen. Programmering på DBHandler begyndt (Nicolai) Programmering på WindowHandler begyndt (Peter)
30/10	Programmering på CinemaGUI begyndt (Nicolai) Programmering på WindowHandler fortsat (Peter)
1/11	Startet programmering af Time klassen (Nicolai) Fortsat programmering på CinemaGUI (Nicolai) og WindowHandler (Peter)
2/11	Færdiggjort Time klassen. Tilføjet SeatStates (Peter) CinemaGUI kan nu generere en biografsal ud fra databasen.
3/11	Programmering af CinemaGUIs eventhandlers er startet Programmering af WindowHandlers eventhandlers er startet NumericTextField tilføjet og bliver brugt i WindowHandler
4/11	Optimering af koden. ShowHolder, DBRow og MovieHolder tilføjet.
7/11	DBSetup skrevet til hurtigt at oprette tabeller og indsætte biografer og shows til den nuværende dag. Mere realistisk data i databasen.
8/11	Programmering af eventhandlers er sluttet.
9/11	CinemaGUI viser nu rækkenumre ud for hver række. Fjernet debug System.out.println. Færdiggjort javadoc.
10/11	Start på rapport: • Fik opsat generelt afsnitslayout • Skrevet brugervejledning • Startet på problemanalyse
11/11	Fortsat rapport: Skrevet på teknisk beskrivelse af programmet. Skrevet på problemanalyse. Omdøbt nogle tabelnavne i databasen og programmet, kontrolleret af programmet fortsat virker som forventet.
14/11	Fortsat rapport: Skrevet forord og indledning. Færdiggjort teknisk beskrivelse af programmet.
15/11	Færdiggjort rapport: • Læst korrektur og rettet stave- og kommateringsfejl • Indsat afprøvning • Indsat projektdagbog i bilaget • Indsat endelige versioner af arbejdsblade i bilaget • Indsat koden i bilaget • Indsat billeder i bilaget
16/11	Udprintning og aflevering af rapport

ENDELIGE VERSIONER AF ARBEJDSBLADE

Databasevalg

Vi har begge erfaring med MySQL og vi så det som et plus at vi ikke behøvede at bruge tid på at lære ny syntax. ITU stiller også en MySQL servere til rådighed, så vi kan forbinde til dem uden at skulle sætte servere op på computere hvor vi afprøver programmet.
Selvom ITU også stiller en Oracle server til rådighed fik kendskabet til syntaksen prioritet.

GUI

Vi har valgt at gøre brugergrænsefladen så simpel som muligt, så vi kan fokusere på problemstillingen og ikke bliver distraheret fra fokusen på at kunne booke sæder hurtigt.
I et professionelt system ved vi at det kan være en nødvendighed for at kunne sælge det, men da dette er en læringsopgave har vi set bort fra dette.

Opbygning af CinemaGUI

Vi valgte af benytte JLabels indeholdende billeder da det ville øge præcisionen af eventhandlers i forhold til tegnede virtuelle felter på et java canvas. Vi mente også at billedikoner ville se pænere ud og give en bedre oplevelse en farvede firkanter, og at tegne mere avancerede grafik via java ville bruge mere tid end vi ville få resultat.

Tid

Vi valgte at benytte Joda time til vores Time class da det var anbefalet flere steder på nettet, og virkede simpelt at arbejde med. Det burde være muligt at genskabe funktionaliteten med java.util.Calendar hvis det bliver nødvendigt.

KILDEKODE

MAIN.JAVA

```
package biobooking;
/**
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
 */

public class Main {

    /**
     * Main class for biobooking.
     * @param args No arguments are currently handled.
     */
    public static void main(String[] args) {
        try{
            new WindowHandler("Biografbooking");
        }
        catch(Exception e){
            System.out.println("Unexpected exception.");
            System.out.println(e);
            System.exit(0);
        }
    }
}
```

DBHANDLER.JAVA

```
package biobooking;
import java.util.ArrayList;
import java.sql.*;

/**
 * DBHandler handles the MySQL connection and queries for the biobooking system.
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
 */
public class DBHandler {
    // The active connection
    private Connection conn;

    /**
     * Initiate the DBHandler by creating a connection.
     */
    private DBHandler(){
        defaultConnect();
    }

    /**
     * The default connection settings. No other settings should be required for testing purposes.
     * @return True upon success, otherwise false.
     */
    private boolean defaultConnect(){
        return mysql connect("mysql.itu.dk", "nbsk", "supersecret", "BGPP cinema");
    }

    /**
     * Check if the connection has timed out. If it has, attempt to reconnect.
     */
    private void maintainConnection(){
        try{
            if(conn.isClosed()){
                System.out.println("Attempting to reconnect.");
                defaultConnect();
            }
        }
        catch(SQLException e){
            System.out.println("maintainConnection() failed: " + e);
```

```
        }
    }

    /**
     * Holder for our simpleton DBHandler.
     */
    private static class DBHandlerHolder{
        private static final DBHandler INSTANCE = new DBHandler();
    }

    /**
     * Get the simpleton DBHandler instance
     * @return The active DBHandler
     */
    public static DBHandler getInstance(){
        return DBHandlerHolder.INSTANCE;
    }


    /**
     * Get all the movie information.
     * @return Return an array of MovieHolders. MovieHolders contain the ID and name of a movie.
     */
    public MovieHolder[] getMovies(){
        maintainConnection();
        String query = "SELECT * FROM `movie` ORDER BY `name`";
        try{
            Statement stmt = conn.createStatement();
            if(stmt.execute(query)){
                ResultSet res = stmt.getResultSet();
                MovieHolder[] movies = new MovieHolder[mysql selected rows(res)];
                int i = 0;
                while(res.next()){
                    movies[i] = new MovieHolder(res.getInt("id"),res.getString("name"));
                    i++;
                }
                return movies;
            }
        }
        catch(SQLException e){
            System.out.println("getMovies() failed: " + e);
        }
        return new MovieHolder[0];
    }

    /**
     * Get the setup of the cinema
     * @param cinemaID The ID of the cinema
     * @return int[rows][seats] of the cinema.
     */
    public int[][] getCinemaArray(int cinemaID){
        maintainConnection();
        int[] rows = getRows(cinemaID);
        int[][] cinema = new int[rows.length][];
        int i = 0;
        for(int row : rows){
            cinema[i] = getSeats(row);
            i++;
        }
        return cinema;
    }

    /**
     * Get a list of shows for the selected movie and date
     * @param movie The id of the movie
     * @param year The selected year
     * @param month The selected month
     * @param day The selected day
     * @return array of ShowHolders. ShowHolders contains the id of the show, the UNIX timestamp for
the start of the show, and the id for the cinema it is displayed in.
     */
    public ShowHolder[] getShows(int movie, int year, int month, int day){
        maintainConnection();
```

```
        String query = "SELECT * FROM `show` WHERE `movie`='"+movie+"' AND
`start`>"+Time.getStartOfDay(day, month, year)+" AND `start`<"+Time.getEndOfDay(day, month, year)+"
ORDER BY `start`";
        try{
            Statement stmt = conn.createStatement();
            if(stmt.execute(query)){
                ResultSet res = stmt.getResultSet();
                ShowHolder[] shows = new ShowHolder[mysql_selected_rows(res)];
                int i = 0;
                while(res.next()){
                    shows[i] = new ShowHolder(res.getInt("id"),
Time.format_Timestamp(res.getInt("start")), res.getInt("cinema"));
                    i++;
                }
                return shows;
            }
        }
        catch(SQLException e){
            System.out.println("getShows() failed: " + e);
        }
        return new ShowHolder[0];
    }

    /**
     * Delete a booking by seat and show
     * @param seat The seat to delete the booking from
     * @param show The show to delete the booking from.
     */
    public void deleteBookingByIDFromSeatAndShow(int seat, int show){
        query("DELETE FROM `booking` WHERE `seat`='"+seat+"' AND `show`='"+show+"'");
    }

    /**
     * Get the rows associated with the cinema
     * @param cinema The cinema ID
     * @return int[] rows for the selected cinema
     */
    protected int[] getRows(int cinema){
        maintainConnection();
        String[] fromRows = {"id"};
        String[] rowsEqualToValues = {"cinema"};
        String[] withValues = {Integer.toString(cinema)};
        return dbRowToInt(mysqlFetchDBRows("row", fromRows, rowsEqualToValues, withValues));
    }

    /**
     * Get the seats associated with the specified row
     * @param row The ID of the row
     * @return int[] of the seats associated with the row.
     */
    protected int[] getSeats(int row){
        maintainConnection();
        String[] fromRows = {"id"};
        String[] rowsEqualToValues = {"row"};
        String[] withValues = {Integer.toString(row)};
        return dbRowToInt(mysqlFetchDBRows("seat", fromRows, rowsEqualToValues, withValues));
    }

    /**
     * Book an array of seats
     * @param show The show ID to book the seats for
     * @param seats The int[] of seat IDs
     * @param phoneNumber The phonenumber to tie the booking to
     */
    public void bookSeats(int show, Integer[] seats, int phoneNumber){
        for(int seat : seats){
            if(!bookSeat(show, seat, phoneNumber)){
                System.out.println("Failed to book seat ID " + seat + "for show ID " + show);
            }
        }
    }

        /**
```

```
     * Add a biobooking order. Expects valid IDs.
     * @param show The show ID
     * @param seat The seat to be booked
     * @param phoneNumber The phonenumber of the person booking.
     */
    public boolean bookSeat(int show, int seat, int phoneNumber){
        return insert("booking", "`show`, `seat`, `phoneNumber`", "'"+show+"', '"+seat+"',
'"+phoneNumber+"'");
    }

    /**
     * Get the cinema id from a certain show.
     * @param show The show id.
     * @return The cinema id upon success, -1 upon failure.
     */
    public int getCinemaFromShow(int show){
        maintainConnection();
        String[] fromRows = {"cinema"};
        String[] rowsEqualToValues = {"id"};
        String[] withValues = {Integer.toString(show)};
        return Integer.parseInt(mysqlFetchDBRows("show", fromRows, rowsEqualToValues,
withValues).get(0).getValue());
    }


    /**
     * Fetch an ArrayList of DBRows
     * @param fromTable The table to fetch rows from
     * @param fromColumns The columns to fetch data from
     * @param columnsEqualToValues The columns to compare to withValues
     * @param withValues The values that columnsEqualToValues must match.
     * @param extra Additional query info. Mainly used for sorting.
     * @return ArrayList of DBRows. DBRows contain the column and the value.
     */
    public ArrayList<DBRow> mysqlFetchDBRows(String fromTable, String[] fromColumns, String[]
columnsEqualToValues, String[] withValues, String extra){
        ArrayList<DBRow> data = new ArrayList<DBRow>();
        try{
            Statement stmt = conn.createStatement();
            if(stmt.execute(buildQueryString(fromTable, fromColumns, columnsEqualToValues,
withValues, extra))){
                ResultSet res = stmt.getResultSet();
                while(res.next()){
                    for(String row : fromColumns){
                        data.add(new DBRow(row, res.getString(row)));
                    }
                }
                return data;
            }
        }
        catch(SQLException e){
            System.out.println("mysqlFetchArray() failed: " + e);
        }
        return new ArrayList<DBRow>();
    }

    /**
     * The same as public ArrayList<DBRow> mysqlFetchDBRows(String fromTable, String[] fromColumns,
String[] columnsEqualToValues, String[] withValues, String extra), but ignoring the optional "extra"
parameter.
     * @param fromTable The table to fetch rows from
     * @param fromColumns The columns to fetch data from
     * @param columnsEqualToValues The columns to compare to withValues
     * @param withValues The values that columnsEqualToValues must match.
     * @return ArrayList of DBRows. DBRows contain the column and the value.
     */
    public ArrayList<DBRow> mysqlFetchDBRows(String fromTable, String[] fromColumns, String[]
columnsEqualToValues, String[] withValues){
       return mysqlFetchDBRows(fromTable, fromColumns, columnsEqualToValues, withValues, "");
    }

    /**
     * Build query strings from String arrays
```

```
     * @param fromTable The table to select data from
     * @param fromColumns The columns to select data from
     * @param columnsEqualToValues The columns to compare to.
     * @param withValues The values that columns must match
     * @param extra Extra optional SQL, for example sorting.
     * @return A formatered query string.
     */
    public String buildQueryString(String fromTable, String[] fromColumns, String[]
columnsEqualToValues, String[] withValues, String extra){
        String selectRows = "";
        for(String row : fromColumns){
            selectRows += "`"+row+"`";
        }
        String requirements = "`"+columnsEqualToValues[0]+"`='"+withValues[0]+"'";
        for(int i = 1; i < withValues.length; i++){
            requirements += " AND `"+columnsEqualToValues[i]+"`='"+withValues[i]+"'";
        }
        return "SELECT "+selectRows+" FROM `"+fromTable+"` WHERE "+requirements+" "+extra+"";
    }

    /**
     * Get the name of the cinema that a show belongs to.
     * @param show The ID of the show
     * @return The name of the cinema that the show belongs to.
     */
    public String getCinemaNameFromShow(int show){
        maintainConnection();
        String[] fromRows = {"name"};
        String[] rowsEqualToValues = {"id"};
        String[] withValues = {Integer.toString(getCinemaFromShow(show))};
        return mysqlFetchDBRows("cinema", fromRows, rowsEqualToValues,
withValues).get(0).getValue();
    }

    /**
     * @param show The show ID. Must be a valid show ID..
     * @return int[] of seat IDs.
     */
    public ArrayList<BookedSeat> getBookedSeats(int show){
        maintainConnection();
        String query = "SELECT * FROM `booking` WHERE `show` = '"+ show +"'";
        ArrayList<BookedSeat> BookedList = new ArrayList<BookedSeat>();
        try{
            Statement stmt = conn.createStatement();
            if(stmt.execute(query)){
                ResultSet res = stmt.getResultSet();
                while(res.next()){
                    BookedList.add(new BookedSeat(res.getInt("seat"), res.getInt("phoneNumber")));
                }
                return BookedList;
            }
        }
        catch(SQLException e){
            System.out.println("getBookedSeats() failed: " + e);
        }
        return new ArrayList<BookedSeat>();
    }

    /**
     * Get the booked phonenumbers for a show
     * @param show The ID of the show
     * @return int[] array of phonenumbers.
     */
    public int[] getBookedPhoneNumbers(int show){
        maintainConnection();
        String[] fromRows = {"phoneNumber"};
        String[] rowsEqualToValues = {"show"};
        String[] withValues = {Integer.toString(show)};
        return dbRowToInt(mysqlFetchDBRows("booking", fromRows, rowsEqualToValues, withValues));
    }

    /**
     * Convert the String values of DBRow to ints.
```

```
     * @param dbRows The ArrayList of DBRows to convert
     * @return int[] array of int values.
     */
    public int[] dbRowToInt(ArrayList<DBRow> dbRows){
        int[] toInt = new int[dbRows.size()];
        int i = 0;
        for(DBRow data : dbRows){
            toInt[i] = Integer.parseInt(data.getValue());
            i++;
        }
        return toInt;
    }

    /**
     * Fast queries that do no not return any results (delete, update, insert, etc).
     * @param query The full query. Must be valid SQL.
     * @return True upon success, false upon failure.
     */
    public boolean query(String query){
        maintainConnection();
        try{
            conn.createStatement().execute(query);
            return true;
        }
        catch(SQLException e){
            System.out.println("query() failed:" + e);
            return false;
        }
    }

    /**
     * Simple INSERT-query
     * @param table The table to insert the data into.
     * @param into The columns to insert data into.
     * @param values The values to insert into the columns.
     * @return True upon success, false on failure.
     */
    public boolean insert(String table, String into, String values){
        return query("INSERT INTO `"+table+"` ("+into+") VALUES ("+values+")");
    }

    /**
     * Get the amount of returned rows (usually for creating new arrays to fit them in).
     * @param rs The resultset to check.
     * @return The amount of results.
     */
    public int mysql_selected_rows(ResultSet rs){
        int result = 0;
        try{
            rs.last();
            result = rs.getRow();
            rs.beforeFirst();
        }
        catch(SQLException e){
            System.out.println("mysql selected rows() failed: "+e);
        }
        return result;
    }

    /**
     * Get the ID of the last inserted row.
     * @param table The table to fetch the ID from.
     * @return The last inserted ID in the table.
     */
    public int getLastID(String table){
        try{
            Statement stmt = conn.createStatement();
            ResultSet result = stmt.executeQuery("SELECT `id` FROM `" + table + "` ORDER BY `id`
DESC LIMIT 1");
            result.next();
            return result.getInt(1);
        }
        catch(SQLException e){
```

```
            System.out.println("getLastID() failed: "+ e);
        }
        return -1;
    }

    /**
     * Connect to a MySQL database.
     * @param server The server connection. For example localhost or mysql.domain.tld.
     * @param user The username for the connection.
     * @param pass The password for the connection.
     * @param database The database to select.
     */
    protected boolean mysql_connect(String server, String user, String pass, String database) {
        try {
            // Fra http://dev.mysql.com/downloads/connector/j/5.1.html -> mysql-connector-java-
5.1.10-bin.jar
            // lagt i C:\Program Files\Java\jdk1.6.0_16\jre\lib\ext
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://"+server+"/"+database+"", ""+user+"",
""+pass+"");
            return true;
        }
        catch (SQLException e) {
            System.out.println("mysql_connect() failed: Kan ikke åbne databaseforbindelsen: " + e);
        }
        catch (ClassNotFoundException e) {
            System.out.println("mysql connect() failed: Can't find the database driver: " + e);
            System.out.println("Please download the file from
http://dev.mysql.com/downloads/connector/j/5.1.html");
            System.out.println("And place it in C:\\Program Files\\Java\\jdk1.6.0_16\\jre\\lib\\ext
(may depend on your java setup and operating system)");
        }
        catch(Exception e){
            System.out.println("mysql_connect() failed: Unexpected exception: " + e);
        }
        return false;
    }

    /**
     * Close the database connection.
     */
    public void close(){
        try{
            conn.close();
        }
        catch(SQLException e){
            System.out.println("close() failed: " + e);
        }
    }
}
```

WINDOWHANDLER.JAVA

```
package biobooking;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.ArrayList;

/**
 * WindowHandler handles the basic graphical user interface, and utilizes the CinemaGUI.
 * @version 14.12
 * @author Peter Ølsted & Nicolai Skovvart
 */
public class WindowHandler  {
    private boolean repaintingWindow = false;
    private int currentShow = 0;
    private static int numberOfOpenWindows = 0;
    private DBHandler dbHandler = DBHandler.getInstance();
    private JFrame frame;
    private static final int[] monthLengths  = Time.getDaysToMonths(2009);
    private static final String[] monthNames = Time.getMonths();
```

```
    private CinemaGUI cinemaGUI = new CinemaGUI(0,0);
    private Container contentPane;
    private JComboBox daysDropDown = new JComboBox();
    private JComboBox monthsDropDown = new JComboBox();
    private JComboBox yearsDropDown = new JComboBox();
    private JComboBox movieDropDown = new JComboBox();
    private JComboBox showsDropDown = new JComboBox();
    private final int YEARS_TO_SHOW = 2;
    private ArrayList<MovieHolder> movieList = new ArrayList<MovieHolder>();
    private final int PHONENUMBER LENGTH = 8; //Danish phone number length
    private NumericTextField phonenumberField = new NumericTextField(PHONENUMBER_LENGTH);
    private JButton findButton = new JButton("Find");
    private JButton bookButton = new JButton("Book");

    /**
     * Initialize a WindowHandler
     * @param windowName The title of the frame.
     */
    public WindowHandler(String windowName){
        setupFrame(windowName);

        contentPane = frame.getContentPane();
        contentPane.setLayout(new BorderLayout());

        Container northMenu = new Container();
        northMenu.setLayout(new FlowLayout());

        updateComponents(northMenu);

        Container southMenu = new Container();
        southMenu.setLayout(new FlowLayout());

        addComponentsToWindow(northMenu, southMenu);

        setEventHandlers();

        setupWindow(true);

        updateShowBox();
    }

    /**
     * Set the preferred size of the window.
     * @return a Dimension with the preferred size.
     */
    private Dimension getPreferredSize(){
        return new Dimension(800, 500);
    }

    /**
     * Set the minimum size of the window
     * @return a dimension with the preferred size.
     */
    private Dimension getMinimumSize(){
        return getPreferredSize();
    }

    /**
     * Update the JComboBox with the amount of daysInMonth
     * @param daysInMonth The amount of days
     */
    private void updateDaysBox(int daysInMonth){
        int selectedItem = daysDropDown.getSelectedIndex();
        daysDropDown.removeAllItems();
        for(int i = 1; i <= daysInMonth; i++){
            daysDropDown.addItem(Integer.toString(i));
        }
        if(selectedItem < monthLengths[monthsDropDown.getSelectedIndex()])
            daysDropDown.setSelectedIndex(selectedItem);
        else
            daysDropDown.setSelectedIndex(daysDropDown.getItemCount()-1);
    }
```

```
    /**
     * Generate the years in the year JComboBox.
     */
    private void updateYearBox(){
        for(int i = Time.getYear(); i < Time.getYear() + YEARS TO SHOW; i++)
            yearsDropDown.addItem(i);
    }

    /**
     * Get the selected month.
     * @return The selected month. Upon failure, return 1.
     */
    private int getSelectedMonth(){
        if(monthsDropDown.getSelectedIndex() > 0)
            return monthsDropDown.getSelectedIndex()+1;
        else
            return 1;
    }

    /**
     * Get the selected day.
     * @return The selected day. Upon failure, return 1.
     */
    private int getSelectedDay(){
        if(daysDropDown.getSelectedIndex() > 0)
            return daysDropDown.getSelectedIndex()+1;
        else
            return 1;
    }

    /**
     * Get the selected year.
     * @return The selected year.
     */
    private int getSelectedYear(){
        return Time.getYear()+yearsDropDown.getSelectedIndex();
    }

    /**
     * Update the showbox. If there is no shows, add error message indicating that there are no
shows.
     */
    private void updateShowBox(){
        repaintingWindow = true;
        showsDropDown.removeAllItems();
        int index = movieDropDown.getSelectedIndex();
        if(index < 0)
            index = 0;

        MovieHolder selectedMovie = movieList.get(index);

        ShowHolder[] currentShows = dbHandler.getShows(selectedMovie.getId(), getSelectedYear(),
getSelectedMonth(), getSelectedDay());

        if(currentShows.length == 0){
            showsDropDown.addItem("Filmen bliver ikke vist denne dag");
            cinemaGUI.removeAll();
            UpdateWindow();
        }else {
            repaintingWindow = false;

            currentShow = currentShows[0].getId();
            for(ShowHolder i : currentShows)
                showsDropDown.addItem(i);
        }
        repaintingWindow = false;
    }

    /**
     * Update the window.
     */
    private void UpdateWindow(){
        frame.pack();
```

```
        frame.repaint();
    }

    /**
     * Update the list of movies.
     */
    private void updateMovieBox(){
        for(MovieHolder film : dbHandler.getMovies()){
            movieDropDown.addItem(film);
            movieList.add(film);
        }
    }

    /**
     * Add names of months to the months JComboBox
     */
    private void updateMonthsBox(){
        for(String month : monthNames)
            monthsDropDown.addItem(month);
    }

    /**
     * Check the length of the phonenumber field. And enable/disable buttons based on this.
     * @return True upon valid phonenumber, otherwise false.
     */
    private boolean checkPhonenumberLength(){
        if(phonenumberField.getText().length() == PHONENUMBER LENGTH){
            findButton.setEnabled(true);
            bookButton.setEnabled(true);
            return true;
        }
        else {
            findButton.setEnabled(false);
            bookButton.setEnabled(false);
            return false;
        }
    }

    /**
     * Return the inserted phonenumber.
     * @return The inserted phonenumber, or 0 if empty.
     */
    private int getPhoneNumberToSearchFor(){
        return phonenumberField.getText().length() > 0 ?
Integer.parseInt(phonenumberField.getText()) : 0;
    }

    /**
     * Set up eventhandlers for all the components.
     */
    private void setEventHandlers(){
        movieDropDown.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                if(!repaintingWindow){
                    updateShowBox();
                }
            }
        });

        yearsDropDown.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                if(!repaintingWindow){
                    updateShowBox();
                }
            }
        });

        monthsDropDown.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                if(!repaintingWindow){
                    updateDaysBox(monthLengths[monthsDropDown.getSelectedIndex()]);

                }
```

```
            }
        });

        daysDropDown.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                if(!repaintingWindow){
                    updateShowBox();
                }
            }
        });

        showsDropDown.addItemListener(new ItemListener(){
            ShowHolder selectedShow;
            public void itemStateChanged(ItemEvent e){
                try{
                    selectedShow = (ShowHolder)(showsDropDown.getSelectedItem());
                    if(selectedShow != null && !repaintingWindow){
                        currentShow = selectedShow.getId();
                        makeNewCinema();
                    }
                } catch(ClassCastException ex){

                }
            }
        });

        phonenumberField.addKeyListener(new KeyListener() {
            public void keyTyped(KeyEvent e){
                if(phonenumberField.getText().length() >= PHONENUMBER_LENGTH)
                    e.consume();

                checkPhonenumberLength();
            }
            public void keyReleased(KeyEvent e){
                checkPhonenumberLength();
            }
            public void keyPressed(KeyEvent e){}
       });

       findButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                makeNewCinema();
            }
        });

        bookButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ArrayList<Integer> bookedSeats = cinemaGUI.getNewBookedSeats();
                dbHandler.bookSeats(currentShow, bookedSeats.toArray(new
Integer[bookedSeats.size()]), Integer.parseInt(phonenumberField.getText()));

                phonenumberField.setText("");
                checkPhonenumberLength();
                makeNewCinema();
            }
        });
    }

    /**
     * Update the cinema
     */
    private void makeNewCinema(){
        contentPane.remove(cinemaGUI);
        cinemaGUI = new CinemaGUI(currentShow, getPhoneNumberToSearchFor());
        contentPane.add(cinemaGUI);
        UpdateWindow();
    }

    /**
     * Setup the frame for the WindowHandler.
     * @param windowName The name of the window
     */
    private void setupFrame(String windowName){
```

```
        numberOfOpenWindows++;
        frame = new JFrame(windowName);

        frame.setDefaultCloseOperation(JFrame.DISPOSE ON CLOSE);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosed(WindowEvent e) {
                numberOfOpenWindows--;
                if(numberOfOpenWindows == 0){
                    dbHandler.close();
                    System.exit(0);
                }
            }
        });
        frame.setPreferredSize(new Dimension(800,370));
        frame.setMinimumSize(new Dimension(800,370));
    }

    /**
     * Setup the window in the middle of the users screen.
     * @param visible The visibility of the window. true for visible, false for not visible.
     */
    private void setupWindow(boolean visible){
        UpdateWindow();
        Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
        frame.setLocation(d.width/2 - frame.getWidth()/2, d.height/2 - frame.getHeight()/2);
        frame.setVisible(visible);
    }

    /**
     * Update all the components of the WindowHandler.
     * @param northMenu The northern menu Container.
     */
    private void updateComponents(Container northMenu){
        contentPane.add(northMenu, BorderLayout.NORTH);
        updateMovieBox();
        updateYearBox();
        updateMonthsBox();
        updateDaysBox(monthLengths[Time.getMonth()-1]);

        daysDropDown.setSelectedItem(Integer.toString(Time.getDate()));
        monthsDropDown.setSelectedItem(monthNames[Time.getMonth()-1]);
        showsDropDown.setPreferredSize(new Dimension(220, 25));

        findButton.setEnabled(false);
        bookButton.setEnabled(false);
    }

    /**
     * Add the components to the graphical user interface
     * @param northMenu The northern menu Container
     * @param southMenu The southern menu Container.
     */
    private void addComponentsToWindow(Container northMenu, Container southMenu){
        northMenu.add(movieDropDown);
        northMenu.add(daysDropDown);
        northMenu.add(monthsDropDown);
        northMenu.add(yearsDropDown);
        northMenu.add(showsDropDown);
        contentPane.add(southMenu, BorderLayout.SOUTH);
        contentPane.add(cinemaGUI);
        southMenu.add(new JLabel("Telefonnummer"));
        southMenu.add(phonenumberField);
        southMenu.add(findButton);
        southMenu.add(bookButton);
    }
}
```

CINEMAGUI.JAVA

```
package biobooking;
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Map;
import java.util.Iterator;

/**
 * CinemaGUI is the graphical display of the cinemas.
 * The cinemas have a variable amount of rows (fetched from the database).
 * The rows have a variable amount of seats (fetch from the database).
 * The seats are able to perform some actions, such as being selected, unbooking, etc.
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
 */
public class CinemaGUI extends JPanel{
    private boolean phoneNumberReservedCurrentSeat = false;
    int show;
    private DBHandler dbHandler = DBHandler.getInstance();
    private HashMap<Integer,SeatStates> changedSeats = new HashMap<Integer, SeatStates>(); //sæde
id, state

    /**
     * Create a new CinemaGUI and display booked seats that match the phoneNumber with the selected
graphic.
     * @param show The show to build the cinema from.
     * @param phoneNumber The phonenumber to check for.
     */
    public CinemaGUI(int show, int phoneNumber){
        this.show = show;
        setLayout(new GridLayout(0, 1));
        makeCinema(phoneNumber);
    }

    /**
     * Make the cinema with the selected phoneNumber
     * @param phonenumber The phonenumber.
     */
    public void makeCinema(int phonenumber){
         if(show == 0){
            return;
        }
        ArrayList<BookedSeat> bookedSeats = dbHandler.getBookedSeats(show);
        phoneNumberReservedCurrentSeat = false;
        int[][] cinemaArray = dbHandler.getCinemaArray(dbHandler.getCinemaFromShow(show));
        int i=cinemaArray.length;
        for(int[] row : cinemaArray){
            JLabel currentRow = new JLabel();
            currentRow.setLayout(new FlowLayout());
            if(i < 10){
                currentRow.add(new JLabel(" "+i));
            }
            else{
                currentRow.add(new JLabel(i+""));
            }
            i--;
            for(int seat : row){
                if(phonenumber != 0){
                    phoneNumberReservedCurrentSeat = findPhoneNumber(bookedSeats, seat,
phonenumber);
                }
                if(phoneNumberReservedCurrentSeat)
                    addSeat("search", currentRow, seat);
                else if(isSeatBooked(seat, bookedSeats))
                    addSeat("booked", currentRow, seat);
                else
                    addSeat("free", currentRow, seat);
            }
            add(currentRow);
        }
        add(new JLabel(Icon("screen")));

        repaint();
    }
```

```
    /**
     * Add a seat to the current row
     * @param type The type of seat. For example "free", "booked", "selected".
     * @param current The current row the seat is being added to.
     * @param id The ID of the seat.
     */
    private void addSeat(final String type, JLabel current, final int id){
        final ImageIcon img = Icon(type);
        JLabel label = new JLabel(img);
        setActionListener(label,type,img,id);
        current.add(label);
    }

    /**
     * Set up actionListeners for the seat
     * @param label The label for the seat
     * @param type The type of the seat
     * @param img The image for the seat.
     * @param id The ID of the seat.
     */
    private void setActionListener(JLabel label, final String type,final ImageIcon img, final int
id){
        label.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                if(img.getImage() == Icon("search").getImage()){
                    img.setImage(Icon("free").getImage());
                    changedSeats.put(id, SeatStates.REMOVED_RESERVATION_SEAT);
                    dbHandler.deleteBookingByIDFromSeatAndShow(id, show);
                }else if(img.getImage() == Icon("free").getImage()){
                    img.setImage(Icon("selected").getImage());
                    changedSeats.put(id, SeatStates.SELECTED SEAT);
                }else{
                    img.setImage(Icon(type).getImage());
                    changedSeats.put(id, SeatStates.REMOVED_RESERVATION_SEAT);
                }
                repaint();
            }
        });
    }

    /**
     * Check if the booked seat belongs to a phonenumber.
     * @param arr The array of BookedSeat's
     * @param seatId The ID of the seat
     * @param phoneNumber The phonenumber to search for
     * @return True if the booked seat belongs to phoneNumber
     */
    private boolean findPhoneNumber(ArrayList<BookedSeat> arr, int seatId, int phoneNumber){
        for(BookedSeat i : arr){
            if(i.seatId == seatId && i.phoneNumber == phoneNumber)
                return true;
        }
        return false;
    }

    /**
     * Quick Icons
     * @param icon The icon to get. For example "free", "Booked", "selected".
     * @return The ImageIcon.
     */
    private ImageIcon Icon(String icon){
        return new ImageIcon(this.getClass().getResource("imgs/"+icon+".jpg"), "");
    }

    /**
     * Set a preferred size for the cinema
     * @return The preferred Dimension
     */
    public Dimension getPreferredSize(){
        return new Dimension(500, 300);
    }
```

```
    /**
     * Set a minimum size for the cinema
     * @return The preferred dimension size
     */
    public Dimension getMinimumSize(){
        return getPreferredSize();
    }

    /**
     * Get the newly booked seats
     * @return ArrayList<Integer> of newly booked seat IDs
     */
    public ArrayList<Integer> getNewBookedSeats() {
        return getSeatsByState(SeatStates.SELECTED_SEAT);
    }

    /**
     * Get all the seats by state
     * @param seatStateToFind The ENUM seatstate.
     * @return ArrayList<Integer> of seats by state
     */
    private ArrayList<Integer> getSeatsByState(SeatStates seatStateToFind) {
        ArrayList<Integer> removedSeats = new ArrayList<Integer>();

        for ( Iterator iter = changedSeats.entrySet().iterator(); iter.hasNext();) {
            Map.Entry entry = ( Map.Entry ) iter.next();

            Integer seatId = (Integer) entry.getKey();
            SeatStates seatState = (SeatStates) entry.getValue();
            if(seatState == seatStateToFind){
                removedSeats.add(seatId);
            }
        }
        return removedSeats;
    }

    /**
     * Check if the seat is booked
     * @param find The seat ID
     * @param in The ArrayList of BookedSeat's
     * @return True if the seat is booked, otherwise false.
     */
    public static boolean isSeatBooked(int find, ArrayList<BookedSeat> in) {
        for(BookedSeat i : in) {
            if(i.seatId == find)
                return true;
        }
        return false;
    }
}
```

TIME.JAVA

```
package biobooking;
import org.joda.time.*;

/*
 * Time is a class with static functions and properties.
 * It is used for initializing the database with testdata, for initializing the of the dropdownmenus
in WindowHandler and for formatting timestamps for queries.
 * @author Nicolai Skovvart & Peter Ølsted
 * @version 14.12
 */
public class Time {
    private static DateTime today = new DateTime();
    private static int thisYear = today.getYear(), thisMonth = today.getMonthOfYear(), thisDay =
today.getDayOfMonth();

    private static String[] months = {"Januar", "Februar", "Marts", "April",
                                      "Maj", "Juni", "Juli", "August",
                                      "September", "Oktober", "November", "December"};
```

```
    /**
     * Get the names of the months
     * @return The list of the names of the months in danish
     */
    public static String[] getMonths(){
        return months;
    }

    /**
     * Get the amount of days the months have for the selected year.
     * @param year The selected year
     * @return int[] array of month lengths.
     */
    public static final int[] getDaysToMonths(int year){
        int[] months = new int[12];
        for(int i = 0; i < 12; i++){
            try{
                DateTime dt = new DateTime(year, i+1, 31, 0, 0, 0, 0);
                months[i] = dt.getDayOfMonth();
            }
            catch(IllegalFieldValueException e){
                try{
                    DateTime dt = new DateTime(year, i+1, 30, 0, 0, 0, 0);
                    months[i] = dt.getDayOfMonth();
                }
                catch(IllegalFieldValueException ex){
                    try{
                        DateTime dt = new DateTime(year, i+1, 28, 0, 0, 0, 0);
                        months[i] = dt.getDayOfMonth();
                    }
                    catch(IllegalFieldValueException exc){
                        System.out.println("getDaysToMonths() try 1 failed: " + e);
                        System.out.println("getDaysToMonths() try 2 failed: " + ex);
                        System.out.println("getDaysToMonths() try 3 failed: " + exc);
                    }
                }
            }
        }
        return months;
    }

    /**
     * Get the current date.
     * @return The current date.
     */
    public static final int getDate(){
        return thisDay;
    }

    /**
     * Create a timestamp today at a specified hour. Used for testdata
     * @param hour The hour to create a show at.
     * @return UNIX timestamp.
     */
    public static final long timestampTodayAtHour(int hour){
        return new DateTime(thisYear, thisMonth, thisDay, hour, 0, 0, 0).getMillis()/1000;
    }

    /**
     * Get the current month.
     * @return The current month
     */
    public static final int getMonth(){
        return thisMonth;
    }

    /**
     * Get the current year.
     * @return The current year.
     */
    public static final int getYear(){
        return thisYear;
    }
```

```
    /**
     * Get the UNIX timestamp for the start of the day of the selected date
     * @param date The date
     * @param month The month
     * @param year The year
     * @return UNIX timestamp for the start of the day for the selected date
     */
    public static final long getStartOfDay(int date, int month, int year){
        return new DateTime(year, month, date, 0, 0, 0, 0).getMillis()/1000;
    }

    /**
     * Get the UNIX timestamp for the end of the day of the selected date
     * @param date The date
     * @param month The month
     * @param year The year
     * @return UNIX timestamp for the end of the day for the selected date
     */
    public static final long getEndOfDay(int date, int month, int year){
        return new DateTime(year, month, date, 23, 59, 0, 0).getMillis()/1000;
    }

    /**
     * Format the timestamp to "hh:mm" format.
     * @param timestamp The timestamp to format
     * @return "hh:mm" format of the timestamp.
     */
    public static final String format_Timestamp(long timestamp){
        DateTime dt = new DateTime(timestamp*1000);
        return ((dt.getHourOfDay()<10) ? "0"+dt.getHourOfDay() : dt.getHourOfDay())
+":"+((dt.getMinuteOfHour()<10) ? "0"+dt.getMinuteOfHour() : dt.getMinuteOfHour());
    }
}
```

SHOWHOLDER.JAVA

```
package biobooking;
/*
 * ShowHolders keep the start time of the show (xx:xx format), the id of the cinema it is being
displayed in, and the ID of the show.
 * @author Nicolai Skovvart & Peter Ølsted
 * @version 14.12
 */

public class ShowHolder {
    private int id;
    private int cinema;
    private String timeOfShow;
    static private DBHandler dbHandler = DBHandler.getInstance();

    /**
     * Initialize the ShowHolder
     * @param id The id of the show
     * @param timeOfShow The time of the show in "xx:xx" format.
     * @param cinema The ID of the cinema the show is displayed in.
     */
    public ShowHolder(int id, String timeOfShow, int cinema){
        this.id = id;
        this.timeOfShow = timeOfShow;
        this.cinema = cinema;
    }

    /**
     * Get the ID of the show
     * @return The ID.
     */
    public int getId(){
        return id;
    }

    /**
```

```
     * Get the ID of the cinema
     * @return The ID of the cinema
     */
    public int getCinema(){
        return cinema;
    }

    /**
     * Get the time of the show
     * @return The time of the show in xx:xx form.
     */
    public String getTimeOfShow(){
        return timeOfShow;
    }

    /**
     * @return The time of the show, followed by the the name of the cinema it is displayed in.
("xx:xx (cinemaname)" format)
     */
    @Override public String toString(){
        return timeOfShow + " (" + dbHandler.getCinemaNameFromShow(id)+")";
    }
}
```

MOVIEHOLDER.JAVA

```
package biobooking;
/*
 * MovieHolders contain the ID and name of the movie
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
 */

public class MovieHolder {
    private int id;
    private String name;

    /**
     * Initialize the MovieHolder
     * @param id ID of the movie
     * @param name Name of the movie
     */
    public MovieHolder(int id, String name){
        this.id = id;
        this.name = name;
    }

    /**
     * Get the ID of the movie
     * @return The ID.
     */
    public int getId(){
        return id;
    }

    /**
     * Get the name of the movie
     * @return The name of the movie
     */
    public String getName(){
        return name;
    }

    /**
     * @return The name of the movie
     */
    @Override public String toString(){
        return name;
    }
}
```

BOOKEDSEAT.JAVA

```
package biobooking;

/**
 * Booked seats have a seat ID and a phonenumber they are booked to.
 * @author Peter Ølsted & Nicolai Skovvart
 * @version 14.12
 */
public class BookedSeat {
    public final int seatId;
    public final int phoneNumber;
    /**
     * Constructor for BookedSeat.
     * @param seatId The seat ID.
     * @param phoneNumber The phonenumber.
     */
    public BookedSeat(int seatId, int phoneNumber){
        this.seatId = seatId;
        this.phoneNumber = phoneNumber;
    }
}
```

DBROW.JAVA

```
package biobooking;

/**
 * DBRows contain the name of the row and the value.
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
 */
public class DBRow {
    private String value, column;

    /**
     * Initialize a DBRow
     * @param column The name of the column
     * @param value The value of the column.
     */
    public DBRow(String column, String value){
        this.column = column;
        this.value = value;
    }

    /**
     * Get the name of the column.
     * @return The name of the column
     */
    public String getColumn(){
        return column;
    }

    /**
     * Get the value of the column.
     * @return The value of the column
     */
    public String getValue(){
        return value;
    }
}
```

NUMERICTEXTFIELD.JAVA

```
package biobooking;
import java.awt.event.*;
import javax.swing.*;
/**
 * NumericTextField is a JTextfield that only accepts digits
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
```

```
 */

public class NumericTextField extends JTextField{
    /**
     * Initialize the textfield
     * @param initValue The initial value of the NumericTextField
     * @param columns The amount of columns the textfield contains.
     */
    public NumericTextField (String initValue, int columns){
        super(initValue, columns);
        this.addKeyListener(new KeyAdapter(){
            public void keyTyped (KeyEvent e){
                if(!keyCheck(e.getKeyChar())){
                    e.consume() ;
                }
            }
        });
    }

    /**
     * Create a NumericTextField with no defaultvalue
     * @param columns The amount of columns the NumericTextField should contain.
     */
    public NumericTextField(int columns){
        this("", columns);
    }

    /**
     * Create a NumericTextField with an int value instead of a string value.
     * @param initValue The int defaultvalue
     * @param columns The amount of columns the NumericTextField should contain
     */
    public NumericTextField(int initValue, int columns){
        this(Integer.toString(initValue), columns);
    }

    /**
     * The check that confirms that it is a valid keypress
     * @param c The typed char
     * @return True if the typed char is valid, otherwise false.
     */
    private boolean keyCheck(char c){
        return ((c == KeyEvent.VK BACK SPACE) || (c == KeyEvent.VK DELETE) ||
(Character.isDigit(c)));
    }
}
```

SEATSTATES.JAVA

```
package biobooking;

/**
 * The ENUM states a seat can have.
 * @author Peter Ølsted & Nicolai Skovvart
 * @version 14.12
 */
public enum SeatStates {
    FREE_SEAT,
    SELECTED_SEAT,
    BOOKED SEAT,
    SEARCH_SEAT,
    REMOVED_RESERVATION_SEAT
}
```

DBSETUP.JAVA

```
package biobooking;
/**
 * DBSetup dumps any old data and creates a few cinemas for testing, as well as movies and shows.
 * @version 14.12
 * @author Nicolai Skovvart & Peter Ølsted
```

```
 */

public class DBSetup {
    private DBHandler db = DBHandler.getInstance();

    /**
     * Setup the database.
     */
    public DBSetup(){
        dropTables();
        createTables();
        createMovies();
        createShows();
        createBigCinema("Galla sal", 6, 24);
        createCinema("Mini bio", 5, 5);
        createCinema("Normal bio", 10, 8);
        createCinema("Funky bio", 2, 25);
        createReversePyramidBio("Omvendt pyramide", 4);
    }

    /**
     * Drop the old tables if they exist.
     */
    private void dropTables(){
        db.query("DROP TABLE IF EXISTS `show`");
        db.query("DROP TABLE IF EXISTS `booking`");
        db.query("DROP TABLE IF EXISTS `movie`");
        db.query("DROP TABLE IF EXISTS `cinema`");
        db.query("DROP TABLE IF EXISTS `row`");
        db.query("DROP TABLE IF EXISTS `seat`");
    }

    /**
     * Create tables.
     */
    private void createTables(){
        db.query("CREATE TABLE `cinema` ( id int(11) NOT NULL auto increment, `name` text NOT NULL,
PRIMARY KEY  (id) ) ENGINE=MyISAM AUTO INCREMENT=0 DEFAULT CHARSET=latin1;");
        db.query("CREATE TABLE `movie` ( id int(11) NOT NULL auto_increment, name text NOT NULL,
duration int(11) NOT NULL default '0', PRIMARY KEY (id) ) ENGINE=MyISAM AUTO_INCREMENT=0 DEFAULT
CHARSET=latin1;");
        db.query("CREATE TABLE `booking` ( id int(11) NOT NULL auto increment, `show` int(11) NOT
NULL, seat int(11) NOT NULL, phoneNumber int(11) NOT NULL, PRIMARY KEY  (id) ) ENGINE=MyISAM
AUTO INCREMENT=0 DEFAULT CHARSET=latin1;");
        db.query("CREATE TABLE `seat` ( id int(11) NOT NULL auto_increment, `row` int(11) NOT NULL,
PRIMARY KEY  (id) ) ENGINE=MyISAM AUTO_INCREMENT=0 DEFAULT CHARSET=latin1;");
        db.query("CREATE TABLE `row` ( id int(11) NOT NULL auto_increment, cinema int(11) NOT NULL,
PRIMARY KEY  (id) ) ENGINE=MyISAM AUTO INCREMENT=0 DEFAULT CHARSET=latin1;");
        db.query("CREATE TABLE `show` ( id int(11) NOT NULL auto increment, movie int(11) NOT NULL,
cinema int(11) NOT NULL, `start` int(11) NOT NULL, PRIMARY KEY  (id) ) ENGINE=MyISAM
AUTO_INCREMENT=0 DEFAULT CHARSET=latin1;");

    }

    /**
     * Create a big cinema with decrementing amounts of seats per row
     * @param name The name of the cinema
     * @param rows The amount of rows
     * @param biggestRow The biggest amounts of seats per row.
     */
    private void createBigCinema(String name, int rows, int biggestRow){
        if(biggestRow-rows*biggestRow >= 0){
            System.out.println("Bad parameters for rows and biggest row. Cannot
createBigCinema();");
            return;
        }
        db.insert("cinema", "`name`", "'"+name+"'");
        int lastCinemaID = db.getLastID("cinema");
        for(int i = 0; i < rows; i++){
            db.insert("row", "`cinema`", "'"+lastCinemaID+"'");
            int lastRowID = db.getLastID("row");
            for(int j = 0; j < biggestRow; j++){
                db.insert("seat", "`row`", "'"+lastRowID+"'");
```

```
            }
            biggestRow = biggestRow-2;
        }
    }

    /**
     * Create a pyramid-shaped cinema
     * @param name The name of the cinema
     * @param rows The amount of rows in the cinema.
     */
    private void createReversePyramidBio(String name, int rows){
        int seatAmount = 1;
        db.insert("cinema", "`name`", "'"+name+"'");
        int lastCinemaID = db.getLastID("cinema");
        for(int i = 0; i < rows; i++){
            db.insert("row", "`cinema`", "'"+lastCinemaID+"'");
            int lastRowID = db.getLastID("row");
            for(int j = 0; j < seatAmount; j++){
                db.insert("seat", "`row`", "'"+lastRowID+"'");
            }
            seatAmount = seatAmount+2;
        }
    }

    /**
     * Create a normal, rectangular cinema
     * @param name The name of the cinema
     * @param rows The amount of rows
     * @param seatsPerRows The amount of seats per row
     */
    private void createCinema(String name, int rows, int seatsPerRows){
        db.insert("cinema", "`name`", "'"+name+"'");
        int lastCinemaID = db.getLastID("cinema");
        for(int i = 0; i < rows; i++){
            db.insert("row", "`cinema`", "'"+lastCinemaID+"'");
            int lastRowID = db.getLastID("row");
            for(int j = 0; j < seatsPerRows; j++){
                db.insert("seat", "`row`", "'"+lastRowID+"'");
            }
        }
    }

    /**
     * Create some movies.
     */
    private void createMovies(){
        db.insert("movie", "`name`, `duration`", "'Ninja Assassin', '99'");
        db.insert("movie", "`name`, `duration`", "'2012', '158'");
        db.insert("movie", "`name`, `duration`", "'The Twilight Saga: New Moon', '128'");
        db.insert("movie", "`name`, `duration`", "'Luftkastellet der blev sprængt', '150'");
        db.insert("movie", "`name`, `duration`", "'(500) Days of Summer', '97'");
    }

    /**
     * Create some shows
     */
    private void createShows(){
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '1',
'"+Time.timestampTodayAtHour(8)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '1',
'"+Time.timestampTodayAtHour(10)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '1',
'"+Time.timestampTodayAtHour(12)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '1',
'"+Time.timestampTodayAtHour(14)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '1',
'"+Time.timestampTodayAtHour(16)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '1',
'"+Time.timestampTodayAtHour(18)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '1',
'"+Time.timestampTodayAtHour(20)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '1',
'"+Time.timestampTodayAtHour(22)+"'");
```

```
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '2',
'"+Time.timestampTodayAtHour(8)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '2',
'"+Time.timestampTodayAtHour(10)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '2',
'"+Time.timestampTodayAtHour(12)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'2', '2',
'"+Time.timestampTodayAtHour(14)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '2',
'"+Time.timestampTodayAtHour(16)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '2',
'"+Time.timestampTodayAtHour(18)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '2',
'"+Time.timestampTodayAtHour(20)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'3', '2',
'"+Time.timestampTodayAtHour(22)+"'");


        db.insert("show", "`movie`, `cinema`, `start`", "'1', '3',
'"+Time.timestampTodayAtHour(8)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '3',
'"+Time.timestampTodayAtHour(10)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '3',
'"+Time.timestampTodayAtHour(12)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '3',
'"+Time.timestampTodayAtHour(14)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '3',
'"+Time.timestampTodayAtHour(16)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '3',
'"+Time.timestampTodayAtHour(18)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '3',
'"+Time.timestampTodayAtHour(20)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '3',
'"+Time.timestampTodayAtHour(22)+"'");

        db.insert("show", "`movie`, `cinema`, `start`", "'4', '4',
'"+Time.timestampTodayAtHour(8)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '4',
'"+Time.timestampTodayAtHour(10)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '4',
'"+Time.timestampTodayAtHour(12)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '4',
'"+Time.timestampTodayAtHour(14)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '4',
'"+Time.timestampTodayAtHour(16)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '4',
'"+Time.timestampTodayAtHour(18)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '4',
'"+Time.timestampTodayAtHour(20)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'1', '4',
'"+Time.timestampTodayAtHour(22)+"'");

        db.insert("show", "`movie`, `cinema`, `start`", "'5', '5',
'"+Time.timestampTodayAtHour(8)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '5',
'"+Time.timestampTodayAtHour(10)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '5',
'"+Time.timestampTodayAtHour(12)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'5', '5',
'"+Time.timestampTodayAtHour(14)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '5',
'"+Time.timestampTodayAtHour(16)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '5',
'"+Time.timestampTodayAtHour(18)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '5',
'"+Time.timestampTodayAtHour(20)+"'");
        db.insert("show", "`movie`, `cinema`, `start`", "'4', '5',
'"+Time.timestampTodayAtHour(22)+"'");

    }
}
```

BILLEDER

imgs/

free.jpg

selected.jpg

booked.jpg

selected.jpg

screen.jpg