

SUBMISSION OF WRITTEN WORK

Class code: GRPRO

Name of course: GRUNDLÆGGENDE PROGRAMMERING

Course manager: DAN HANSEN

Course e-portfolio: GRPRO-AUTUMN 2014

Thesis or project title: KAFFEKLUBBEN

Supervisor: JESPER MADSEN

Full Name: Birthdate (dd/mm/yyyy): E-mail:

1. MARK ROSTGAARD MORTENSEN 09/04-1993 MROM@itu.dk

2. AMANDA ENGEL THILO 21/03-1993 AENT@itu.dk

3. FREDERIK HOVMAND JØRGENSEN 20/01-1993 FHOV@itu.dk

4. _____ @itu.dk

5. _____ @itu.dk

6. _____ @itu.dk

7. _____ @itu.dk



EXAM PROJECT
Biobooking

KAFFEKLUBBEN

Amanda Engel Thilo

Frederik Hovmand Jørgensen

Mark Rostgaard Mortensen

DECEMBER 17, 2014

Den rapport er udarbejdet i December 2014 i et fire ugers projekt på IT-Universitetet i København under vejledning af Jesper Madsen. I projektet udviklede vi et softwaresystem til booking i biografer skrevet i Java. Programmet har til opgave at skulle assisterer en ekspedient i personens arbejdsopgaver der har med kundehenværdelser af gøre, f.eks: Book nye pladser eller slette en tidligere bestilling. Programmet skal ikke kunne håndtere salg.

Indhold

1	Indledning	4
1.1	<i>Baggrund</i>	4
1.2	<i>Problemstilling</i>	4
1.2.1	Krav til programmet	4
1.2.2	Brugerscenario	4
1.2.3	Systemdesign	5
2	Problemanalyse	6
2.1	<i>Vores løsning</i>	6
2.1.1	Build Reservation	6
2.1.2	Alternative løsninger	7
2.2	<i>Databasedesign</i>	7
3	Brugervejledning	8
3.1	<i>Programmet</i>	8
3.1.1	Begrænsninger	10
3.1.2	Fejlmeddelelser	10
3.2	<i>Eksempel</i>	10
4	Teknisk Analyse	11
4.1	<i>Model View Controller</i>	11
4.1.1	Moduler	11
4.1.2	Datastrukturer	11
4.1.3	Algoritmer	11
4.2	<i>Brugergrænsedesign</i>	12
4.2.1	Begrænsninger	12
5	Afprøvning	13
5.1	<i>Brugerafprøvning</i>	13
5.2	<i>Unit test</i>	13
5.3	<i>Resultat</i>	14
6	Konklusion	15
7	Litteratur	16
	Figurer	17
8	Bilag	18

Kapitel 1

Indledning

1.1 Baggrund

Det problemområde vi har arbejdet med i forbindelse med vores projekt, er udviklingen af et softwaresystem til brug for en ekspedient der betjener en billetluge i en biograf. Det område der skulle dækkes var reservation både når kunden stod foran billetlugen, samt når kunden ringede ind på telefon. Vores løsning skulle håndtere ekspedientens arbejdsopgaver i sådanne situationer. Løsningen skal derfor fokusere udelukkende på reservationer, og ikke salg.

1.2 Problemstilling

Den overordnede problemstilling som vores program besvarer, er hvordan man udviklinger et simpelt, brugervenligt og databasebaseret softwaresystem til brug for en ekspedient i en billetluge.

1.2.1 Krav til programmet

Af krav til programmet kan nævnes at der i forbindelse med reservationerne skal oplyses navn eller anden identitet på kunden. Vi har i vores program valgt at den centrale information omkring brugeren er dennes telefonnummer. Det har vi gjort af den grund af flere kunder godt kan have samme navn, men at ens telefonnummer er unikt.

Derudover skal biografen indeholde flere sale, og sende flere forskellige forestillinger i løbet af dagen. Salenes størrelse, antal pladser mm. skal fremgå af databasen frem for at være indkodet i selve programteksten. Derfor har vi i stedet for at skrive data ind i programteksten, valgt at lave en speciel klasse til at hente data fra databasen.

Det samme er gældende for de enkelte forestillinger, der derfor også bliver gemt i databasen.

1.2.2 Brugerscenario

- skal laves om Når ekspedienten åbner vores biobooking-system vil der på venstre side af vinduet være opelistet de film der spilles i biografen i øjeblikket. Klikker ekspedienten på en af disse film vil der på højre side af vinduet vise sig de tilhørende spilletider til den valgte film. Klikker ekspedienten derimod ikke på noget, vil der stå følgene tekst *Klik på en film til venstre for at vise tidspunkter*. Øverst oppe i vinduet vil man se tre faner: *Forestillinger, Reservation og Ret reservation*. Man kan godt navigere mellem fanerne, men har man ikke først valgt film og tidspunkt vil *Reservation* være ubrugelige.

Efter at ekspedienten har valgt en film og en spilletid vil systemet automatisk skifte videre til den næste fane: *Reservation*. Her kan ekspedienten vælge frit mellem de ledige sæder. Dette gøres ved at ekspedienten enten klikker enkeltvis på de grønne sæder, eller ved at trække musen

over de ønskede sæder. De valgte sæder vil nu blive blå, så ekspedienten kan se hvilke der er valgt. Er sæderne tilfredsstillende skal ekspedienten skrive navn samt telefonnummer på kunden ind i et informationsfelt nedernst til højre på siden. Nedderst på siden ses i venstre hjørne antallet af sæder i alt, samt det ledige antal sæder. Dette kan ekspedienten også se på det grafiske billede over biografsalen, da de allerede reserverede sæder er røde, samt umulige at vælge for reservation. Efter kundens information er indtastet og ekspedienten har klikket på *Fuldfør reservation* kommer et pop-up vindue op, hvor der står skrevet: *Bestillingen er gennemført*.

Ønsker den pågældende kunde derimod at ændre eller slette en reservation, skal ekspedienten klikke sig ind på den sidste fane: *Ret reservation*. På denne side finder ekspedienten øverst oppe et input felt hvor kundens telefonnummer skal indtastes. Når ekspedienten har klikket enter vil kundens reservation(er) dukke op nedenfor i en pæn liste. Ekspedienten kan så klikke på den reservation som kunden ønsker ændret. Nu føres ekspedienten til en popup *Reservation* side, hvor de sæder tilknyttet den pågældende reservation nu ses med blåt. Ekspedienten kan således tilføje eller fjerne sæder ved at klikke på sæder med musen og derefter klikke på *Ret reservation*. Ønsker kunden helt at slette reservationen kan dette lade sig gøre ved at klikke på *Slet reservation*.

1.2.3 Systemdesign

Vores system er overordnet baseret på data fra databasen. Systemet generer layout ud fra forskellige data i databasen. Dette indebærer bl.a. hvilke film der er i film-tabellen, hvilke forestillinger der er tilknyttet til en given film, hvilke reservationer der er tilknyttet en given forestilling, hvilke reservationer der er tilknyttet et givent telefonnummer mv. Programmet er delt op, så mest mulig databehandling foretages hos klienten/brugeren. Dette indebærer objekter med informationer om de forskellige forestillinger, arrays med valgte sæder mv. Databaseforespørgsler er lavet så omfattende som muligt, så der udføres så få forespørgsler som muligt og data gemmes i objekter og arrays eller bliver benyttet med det samme i den grafiske fremstilling af programmet som eksempelvis ved generering af film-knapper.

Kapitel 2

Problemanalyse

2.1 Vores løsning

Den overordnede problemstilling vi har valgt at fokusere på, er hvordan man udvikler et simpelt og implecit program der både og brugervenligt og hvor databasen opbevarer det meste data.

Vores program løser problemstilligen ved at have alt information omkring forestillinger, sale, film og reservationer gemt i en database. Dette gør selve koden nem at vedligeholde, og vi har på den måde undgået unødvendig kodeduplikering. For at opfylde ønsket om brugervenlighed har vi valgt at vise programmet i ét vindue, hvor brugeren så nemt kan navigere, ved hjælp af tabs, imellem de forskellige funktionaliteter programmet har.

2.1.1 Build Reservation

En af de større udfordringer, som man også har kunne tackle på en masse forskellige måder, var når biografalen skulle bygges grafisk. Den primære metode vi brugte her var buildReservationsScene(int showID) i vores Controller.java class. Alt informationen omkring biografalen får vi fra vores database, dette bliver behandlet i metoden, her er det bl.a. det forskellige labels der bliver sat (cinemaNameLabel, movienameLabel etc). Med informationen fra databasen kører vi et for-loop der kører igennem kolonerne, og inden i for loopet kører der endnu et for-loop der kører sæderne igennem for hver kolonne. Det er rimelig lige til og i højgrad den bedste måde vi kunne komme på at gøre det. Sæderne bliver grafisk vist som firkanter, her kunne man godt have lavet det som knapper. Vi valgte at bruge Ractangle objekter fordi det var nemt at manipulere det udseende vi gerne ville give dem uden at vi behøvede at gå på kompromis med funktionaliteten.

```

for(Antal kolonner){
    for(Antal rækker på kolonne){
        if(Sæde er reserveret){
            Gør sæde rød
        }
        else{
            gør sæde grøn
        }
    }
}

```

Til sidst er der måden du reserverer flere pladser. En oplagt mulighed ville være at have en dropdown menu i bunden af vinduet, der, når den klikkes, viser en liste fra med tal fra 1 til antallet af frie sæder (hvis der er nogle frie sæder tilbage). Denne mulighed er ganske fin, vi besluttede os dog for at gå med noget mere interaktivt hvor ekspedienten blot skal trække musen henover de sæder der skal vælges for at vælge dem. Dette er en feature der gør det nemmere for ekspedienten at vælge en masse sæder på engang og samtidig super fleksibel.

2.1.2 Alternative løsninger

An alternativ måde at løse problemstillingen på ville være at fokusere på andre måder at booke billetter på. Vi har valgt at man først skal vælge film, derefter tid og til sidst sæder. Man kunne have valgt at gøre det muligt for brugeren at vælge dato eller tid før valg af film. Vi har dog valgt den løsning vi syntes var mest brugervenlig. En anden mulighed er at man kunne have lavet en liste der viste alle forestillinger sorteret efter spilletid - uden at tage hensyn til sortering i forhold til film. Denne mulighed syntes vi designmæssigt ville fungere meget rodet. Derudover kunne vi have valgt at gemme vores reservationer i programkoden frem for i databasen. Vi diskuterede dette i begyndelsen af projektet, men fandt det for besværligt at arbejde med, da det ville kræve større mængder kode, fylde mere plads samt resultere i at vores program formentlig ville bruge længere tid på at compile og køre.

2.2 Databasedesign

En af de to overordnede tabeller i vores database er *cinemas* som indeholder information omkring vores biografsale. Dette omfatter salens navn, id, antal rækker og antal sæder i hver række. Vores sæder er ikke repræsenteret som en tabel i vores database, men derimod gemt som et 2 dimentzionelt array.

Den anden vigtige tabel er *movies* som indeholder information omkring de film der går i biografen. Det eneste information denne tabel holder er navn og id.

I begge tabel er det, det pågældende id som fungerer som nøgle. Altså `sal_id` og `movie_id`.

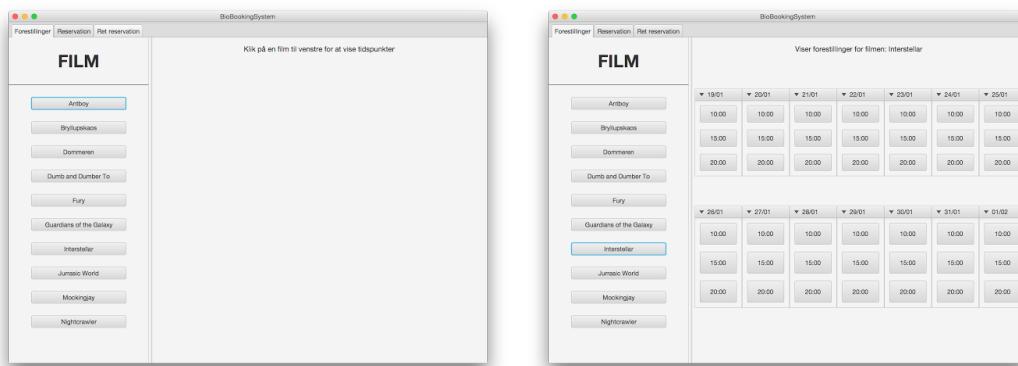
De to tabeller *movie* og *cinema* bliver koblet sammen i tabellen *show*. Her bruges deres nøgler til at sammensætte de forskellige forestillinger som vores biograf kan vise. I *shows* tilføjes så et nyt id til de enkelte forestillinger samt et timestamp - som er det tidspunkt forestillingen skal begynde.

Kapitel 3

Brugervejledning

3.1 Programmet

Programmet er designet med 3 tabs i toppen. *Forestillinger*, *Reservation* og *Ret reservation*. Når programmet åbnes starter brugeren i *Forestillinger*-vinduet og der vises en liste i venstre side med de film som kører i biografen. Brugeren klikker på den film, som kunden ønsker at reservere billetter til. Når en film vælges bliver tiderne for filmforestillinger vist for den pågældende film.



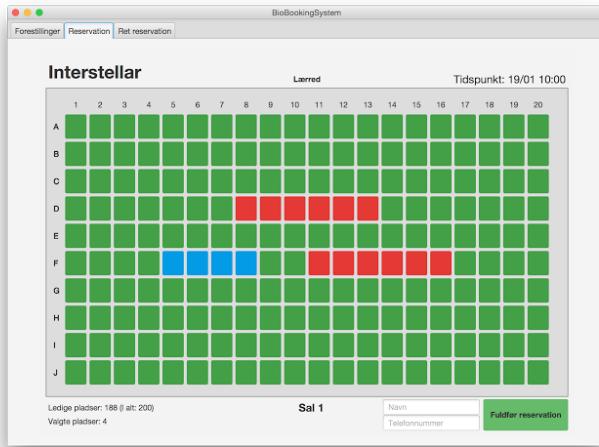
Figur 3.1: Forestillinger-vinduet før og efter en film er valgt

Klikker ekspedienten derimod ikke på noget, vil der stå følgene tekst *Klik på en film til venstre for at vise tidspunkter*, og har man ikke først valgt film og tidspunkt vil *Reservation* være ubrugelige.

Efter at have valgt film, vælger brugeren tidspunktet, som kunden ønsker at reservere til. Når en forestilling vælges bliver brugeren videreført til næste tab - altså reservation. Her vises et vindue med firkanter som illustrerer sæderne i biografen. Ledige sæder er illustreret som en grøn firkant og optagede sæder er illustreret som røde sæder. Brugeren klikker på de sæder som kunden ønsker at reservere - brugeren kan desuden holde musen nede og hove musen over de sæder som skal vælges. De valgte sæder skifter farve til blå og brugeren kan se hvor mange sæder vedkommende har markeret nederst i venstre hjørne.

Når sæderne er valgt skrives kundens navn og telefonnummer ind i tekstmærkerne nederst i højre hjørne. Afslutningsvist trykkes der på knappen *Fuldfør reservation*.

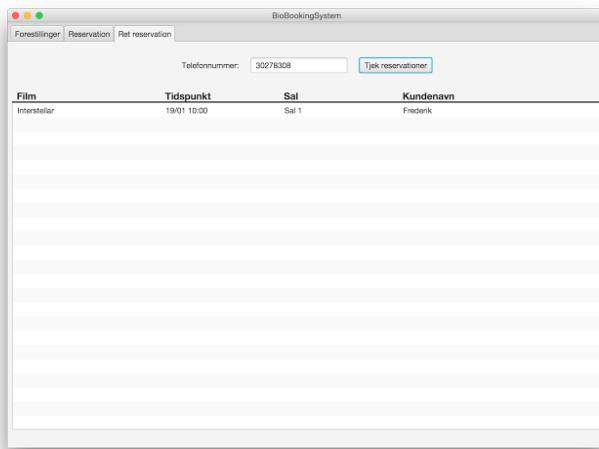
Hvis kunden fortryder sin bestilling eller har ændringer til en eksisterende bestilling, så klikker brugeren på tabben *Ret reservation* i toppen af programmet. Brugeren bliver taget til et vindue hvor kundens telefonnummer skal indtastes. Når telefonnummeret er indtastet og der er trykket på Enter-tasten eller på *Tjek reservationer*-knappen, så vises en liste nedenunder med de



Figur 3.2: Biograf sal med reserverede og valgte sæder

reservationer som kunden har tilknyttet til sit telefonnummer. Der vises kun reservationer for forestillinger som ikke er blevet vist - gamle reservationer vises altså ikke. Brugeren dobbeltklikker på den reservation der skal rettes og et nyt vindue åbner som ligner reservationsvinduet. De optagede sæder er røde, de ledige er grønne og de sæder som er tilknyttet til den pågældende reservation vises som blå. Brugeren kan tilføje eller slette sæder. Et valgt er blåt og markeringen fjernes ved at klikke på sædet igen. Når ændringen til reservationen er valgt, så klikker brugeren på *Rediger reservation*. Skal reservationen derimod slettes helt, så klikker brugeren på *Slet reservation*.

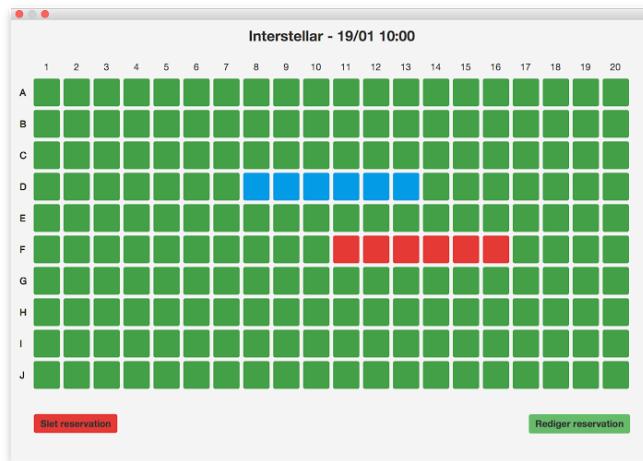
Hvis kunden fortryder sin bestilling eller har ændringer til en eksisterende bestilling, så klikker brugeren på tabben *Ret reservation* i toppen af programmet. Brugeren bliver taget til et vindue hvor kundens telefonnummer skal indtastes. Når telefonnumret er indtastet og der er trykket på Enter-tasten eller på *Tjek reservationer*-knappen, så vises en liste nedenunder med de reservationer som kunden har tilknyttet til sit telefonnummer.



Figur 3.3: Forestillinger-vinduet før og efter en film er valgt

Der vises kun reservationer for forestillinger som ikke er blevet vist - gamle reservationer vises altså ikke. Brugeren dobbeltklikker på den reservation der skal rettes og et nyt vindue åbner

som ligner reservationsvinduet.



Figur 3.4: Vinduet hvor brugeren retter sin reservation

De optagede sæder er røde, de ledige er grønne og de sæder som er tilknyttet til den pågældende reservation vises som blå. Brugeren kan tilføje eller slette sæder. Et valgt er blåt og markeringen fjernes ved at klikke på sædet igen. Når ændringen til reservationen er valgt, så klikker brugeren på *Rediger reservation*. Skal reservationen derimod slettes helt, så klikker brugeren på *Slet reservation*.

3.1.1 Begrænsninger

3.1.2 Fejlmeddelelser

3.2 Eksempel

Kapitel 4

Teknisk Analyse

4.1 Model View Controller

4.1.1 Modularer

Programmet har 2 klasser: DBConnect og buildHolder. DBConnect foretager alt der har med databasen at gøre. Dette er eksempelvis databaseforespørgsler hvor alle film returneres i et LinkedHashMap, hvor data såsom reservationer, salstørrelse mv. for en given forestilling hentes, hvor data såsom reservationer indsættes eller opdateres i databasen. Den væsentligste del af DBConnect-klassen må være funktionen getCon, der sørger for at der er en gyldig forbindelse til databasen.

```
private Connection getCon() {
    try {
        if(!con.isValid(30)) {
            con = DriverManager.getConnection("jdbc:mysql://mysql.itu.dk:3306
                /kaffeklubben", "kaffeklubben", "pass");
        }
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
    return con;
}
```

En anden klasse er *buildHolder*. Et *buildHolder*-objekt lagrer data for en given forestilling. Der bruges getters og setters, så de data der lagres også kan hentes efterfølgende. *buildHolder* objektet bruges efterfølgende til at bygge reservationssalen med data som filmnavn, tidspunkt for forestillingen, størrelsen på salen (rows/columns), reserverede sæder mv.

Det er *controlleren* der gør brug af disse objekter, når data skal visualiseres og altså fremvises for brugeren.

4.1.2 Datastrukturer

4.1.3 Algoritmer

Når et DBConnect objekt initialiseres bliver der oprettet og gemt en connection til databasen i et felt. Funktionen *getCon* tester om denne forbindelse stadig er gyldig. Hvis den ikke er gyldig, så oprettes der en ny gyldig forbindelse og denne returneres. Derved kan forbindelsen eksempelvis kaldes som *getCon().createStatement()*; og altid bruge en gyldig forbindelse.

getMovies tager ingen parametre og returnerer et LinkedHashMap med filmens id som key og filmens navn som value. Sorteret alfabetisk efter filmnavn (A-Z).

getMovieSchedule tager film id som parameter og returnerer et LinkedHashMap med show id som key og tidspunktet for forestillingen (timestamp) som value. Data er hentet fra databasen med en sql-query hvor film id'et er det ene where-parameter og tiden for forestillingen er større/senere

end det nuværende tidspunkt. Hele forespørgslen er sorteret med den nærmeste forestilling først - altså *time* ASC.

getBuildSceneInfo tager et forestillings id som parameter og laver et *buildHolder* objekt med information om forestillingen. Først hentes generelt information såsom filmnavn, tidspunkt for forestillingen, størrelsen på salen (højde og bredde) m.fl. Efterfølgende hentes alle de reserverede sæder til den pågældende forestilling. Disse reservationer gemmes i et multidimensional array $[x][y]$ af typen *Boolean*, hvor værdien i det ydre array er x-koordinaten til sædet og værdien i det indre array er y-koordinaten. Et reserveret sæde sættes til at være *true* på sædets plads - eksempelvis vil sæde 5:7 være $[5][7] = \text{true}$, hvis det er reserveret.

insertReservation er den metode, som sørger for at de valgte sæder og tilknyttede kunde-informationer indsættes i databasen. Der tages fire parametre: en *ArrayList* af typen string med sæderne, en *Integer* med forestillingens id, en *String* med kundens navn og ligeledes en med kundens telefonnummer. Sæderne gemmes som string med et kolon som splitter mellem sædets x- og y-værdi fx 5:8.

Først indsættes kundens navn, telefonnummer og forestillingens id i tabellen *reservations*. Derefter køres funktionen *getLastReservationId*, som returnerer senest indsatte reservations id. Dette id skal nemlig bruges til at linke de valgte sæder med reservationen. Dernæst løbes vores *ArrayList* igennem for at indsætte hvert sæde i reservation ind i vores database.

```
for(String seat : seats) {  
    query = "INSERT INTO reservationlines (reservation_id, seat_x, seat_y) VALUES ('" +  
        lastid + "', '" + seatInfo[0] + "', '" + seatInfo[1] + "')";  
}  
_____
```

Koden ovenfor illustrerer hvordan vi indsætter seat i databasen. Vores sæde-string skal splittes op, så vi kan indsætte en x-værdi og en y-værdi i databasen. Man splitter vores string op i et *string-array* hvor index 0 er alt tekst før første kolon. Index 1 er alt tekst før andet kolon osv.

```
String[] seatInfo = seat.split(":");  
_____
```

Derved kan vi nu tilgå vores x-værdi som *seatInfo[0]* og vores y-værdi som *seatInfo[1]*. Hvis hele metoden forløber fejlfrit, så returneres der boolean *true*. Hvis der forekommer en *exception* så returneres *false*.

4.2 Brugergrænsedesign

4.2.1 Begrænsninger

Kapitel 5

Afprøvning

5.1 Brugerafprøvning

5.2 Unit test

Vores klasse *DBConnectTest* er en testklasse indeholdende tests. Den primære funktion for disse test er at teste om vores programtekst snakker rigtigt sammen med databasen. Fordi vi primært tester koden i relation til databasen er det ikke rene unit test vi udfører. Faktisk er det det man kalder integration tests. Integration test er test der bruger en database, et netværk eller et andet eksternt system som fx en mailserver. Typisk for en integration test er også at den udfører I/O. I vores tilfælde tager vores tests input fra en database og udskriver et grafisk output til skræmmen.

Testen *testGetMovies()* tester om film og ide passer sammen. Dette gøres ved at vi giver testen et forventet uddata, som er bestemte id's tilknyttet bestemte film. Når testen kører, testes derfor om det forventede uddata passer overens med det faktiske uddata.

testTimeStamp() tjekker hvorvidt en given films forestillinger bliver sorteret korrekt efter tid. Det gøres ved at lave et while-loop som kører igennem alle tidspunkterne og tjekker om det nuværende timestamp er lavere end det foregående.

For at tjekke om et reserveret sæde vise med rød farve, og dermed ikke har nogle funktioner. Sædernes funktioner er nemlig tilknyttet deres farve, således at de røde sæder ikke er mulige at klikke på. *testReservedSetColor()* henter et reserveret sæde og derefter sætter dens farve lig med den forventede røde farve.

testInsertReservation() tester om metoden *InsertReservation()* rent faktisk opretter en reservation.

```
@Test
public void testInsertReservation(){
    ArrayList<String> seats = new ArrayList<String>();
    seats.add("1:2");
    dbConnect.insertReservation(seats, 120, "Amanda", "26802103");
    dbConnect.getLastReservationId();
```

Ovenfor ses en del af testmetoden *testInsertReservation*. I metoden begynder vi med at oprette en *ArrayList* til sæder, og herefter indsætter vi et sædet i Array'et som har plads [1:2]. Tredje linje kalder *insertReservation* metoden fra *dbConnect* klasse, og giver metoden de værdier den skal bruge for at oprette en reservation. Den sidste linje i *testInsertReservation* bruger metoden *getLastReservation* metoden, igen fra *dbConnect*, til at hente det sidste oprette reservations id - som er den reservation vi lige har oprettet i linjen ovenover.

Den sidste testmetode *testGetReservation* tester metoden *getReservation*. Dette gøres ved at testen først tjekker om en reservation med telefonnummeret 11223344 eksisterer, og returner *true* hvis reservationen ikke eksisterer. Herefter opretter testen selv en reservation under det samme telefonnummer, og til sidst tjekker testen igen om der eksisterer en reservation med det givne telefonnummer. Denne gang skal testen returnere *true* hvis der eksisterer en reservation.

Både *testInsertReservation* og *testGetReservation* har en *@After* test, som sletter de reservationer der bliver lavet i de to tests. Dette gøres ved at *@After* kalder *deleteReservation* fra *dbConnect*.

5.3 Resultat

Kapitel 6

Konklusion

Kapitel 7

Litteratur

Figurer

- 3.1 Forestillinger-vinduet før og efter en film er valgt 8
- 3.2 Biograf sal med reserverede og valgte sæder 9
- 3.3 Forestillinger-vinduet før og efter en film er valgt 9
- 3.4 Vinduet hvor brugeren retter sin reservation 10

Kapitel 8

Bilag