**TDT4100 Objektorientert programmering**

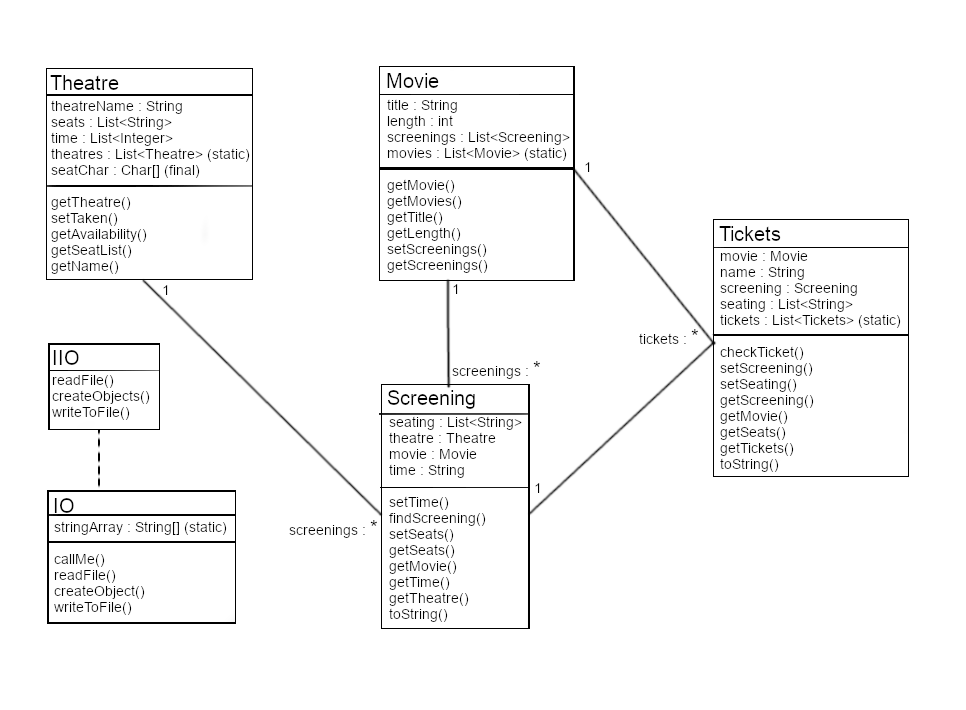
Prosjektdokumentasjon

**Prosjektbeskrivelse:**

Jeg har laget en applikasjon som simulerer at du kjøper billetter til en filmvisning på kino. Det er er lagt inn fire ulike filmer, fire ulike saler og fire ulike tidspunkt for hver

film; hvor én tilhører per sal. Applikasjonen åpner til en side hvor man kan velge hvilken film man ønsker billett til, eller se dine allerede valgte billetter. Deretter kan man velge hvilke tidspunkter man ønsker å se filmen hvor det da også velges én av salene. På neste side kan man velge hvor mange plasser man ønsker i salen, og billetten knyttes til et navn man skriver inn i feltet der. Når man bytter til neste side reserveres billetten og du får en liste over hvilke plasser du har fått, samt informasjon om visningen. Listen over billetter lagres i en tekstfil som må slettes manuelt dersom man ønsker å fjerne billetter. Det er ikke mulig å booke flere billetter til samme visning med samme navn, men det kan gjøres med ulike navn eller til ulike visninger på samme film. Til informasjon så lages filmene og visningene ut i fra et dokument i samme mappe som billettene, dersom man ønsker å endre hvilke navn og så videre kan dette gjøres der i ‘cinema.txt’. Videre lagres de bestilte billettene (som tar opp en liste i programmet) i tickets.txt; denne må tømmes dersom man ønsker å fjerne en eller alle billettene.

**Klassediagram:**

****

*\*Usikker på hvordan IIO interfacet og IO skal føres opp på klassediagrammet.*

**Spørsmål:**

I prosjektet dekkes det en rekke av teorien fra pensum. Det er flere interakterende klasser i tillegg til klassene: ‘Controller’ og ‘App’ som kreves for FXML delen av prosjektet. Fra pensum brukes det blant annet ‘Interface’ for å opprette en ‘IO’-klasse som leser og skriver fra to ulike filer, ‘interfacet’ kunne derimot vært brukt til å lage to ulike klasser fremfor én felles. I mellom klassene deles mange oppgaver, det er kall på kryss og tvers for å binde alt sammen; Noen oppgaver delegeres for eksempel vekk fra ‘Screenings.java’ til ‘Movie.java’ ettersom oppgaven gjerne er knyttet til spesifikke filmer. For å finne nødvendige objekter i ulike lister tas det i bruk ‘lambda streams’ for å søke etter disse. Naturligvis etter prosjektkravene benyttes det ulike former for feilhåndtering i systemet. Validering, ‘try/catch’ og ‘exceptions’ brukes for å sikre at metoder og funksjoner brukes som de er tiltenkt slik at ingenting slår feil. I tillegg blir de fleste metoder testen for seg selv ved bruk av ‘JUnit’-tester.

I applikasjonen tas det ikke i bruk for eksempel observatør prinsippene, noe som kunne gjort Controlleren enklere å lese og oppgaver kunne vært flyttet ut av denne til en dedikert observatør som oppdaterer for eksempel billetten i stedet. Jeg tar heller aldri i bruk iteratorer selv om det nok kunne blitt tatt i bruk for å finne objekter på samme måte lambda streams blir brukt i dette prosjektet. Ei brukes arv i løpet av prosjektet, dog det er lite bruk for i utgangspunktet. Én mulighet kunne vært å arve visninger fra salen for et tettere bånd mellom disse to klassene, men virker ikke særlig nødvendig for dets bruk slik den står nå. Til slutt tas det heller ikke bruk i abstrakte klasser.

Applikasjonen er bygget opp etter model-view-controller-prinsippet og tar i bruk dette blant annet ved at én controller klasse tar seg av interaksjonen med brukergrensesnittet; og da binder brukergrensesnittet sammen med resten av koden. I ettertid ser jeg at noen av assosiasjonene i applikasjonen kunne vært bedre delegert ettersom det er noen feller egenskaper i bildet her. I tillegg hadde nok bruk av observatør-prinsippene vært ideelle å ta i bruk her fremfor kall slik det står nå.

Når det kommer til tester skal alle metodene være testet med JUnit, ikke minst er de alle testet med at programmet fungerer som det er tiltenkt, uten noen kryss og feil under kjøring. Har derimot valgt å ikke teste de få metodene i IO.java for å forhindre kluss når det kommer lagret informasjon som brukes av selve programmet, dermed forhindrer dette at det oppstår falske billetter, eller at filer blir tomme. Metodene er som sagt testen grundig i utviklingen av selve programmet og disse metodenes tenkte bruksområdet.

Dersom jeg skulle startet utviklingen av dette prosjektet på nytt ville jeg definitivt hatt mer informasjon lagret om filmene og visningene på de respektive sidene. Det hadde blitt mer realistisk for programmet og ha informasjon om for eksempel produsenter og språk enn slik det står nå. I tillegg hadde observatør/observert-prinsippene vært særlig mye ‘penere’ å bruke fremfor kall som nevnt tidligere. Fremfor ‘lambda streams’ kunne nok disse funksjonene tatt i bruk iterering for å åpne samme funksjonalitet. En mer grundig delegering hadde også definitivt vært mer gjennomtenkt dersom programmet skulle bli laget på nytt ettersom jeg ikke er fornøyd med slik dette ble. Generelt sett mener jeg programmet hadde endt desto bedre dersom jeg visste om flere av teknikkene vi lærte senere i pensum, da som observatører, iterering og delegering. Funksjoner som ‘callMe’ i ‘IO’ hadde jeg nok droppet ettersom denne er særlig unødvendig, og er mer tilstede for å vise kunnskap om variable mengder input på en metode. Til slutt tror jeg kompetansen jeg har lært nå ville gitt bedre resultater ettersom jeg nå er mer vandt med språket, og hvilke teknikker som kan brukes.

I løpet av utviklingen var det særlig ett problem, jeg fant ingen muligheter for å få bilder til fungere som tiltenkt i programmet når det kjører. Det ble utprøvd flere ulike måter å legge til filmplakater for en mer virkelighetsnær opplevelse av brukergrensesnittet, men her fant jeg ingen sikker løsning som fungerte konsist med FXML. Er for så vidt usikker om ‘initialize’ metoden er beste løsning for ønsket oppførsel i ‘Controller.java’, særlig da kommentert med ‘switch/case’ som jeg valgte å bruke.