Dokumentasjon for HotelBooking

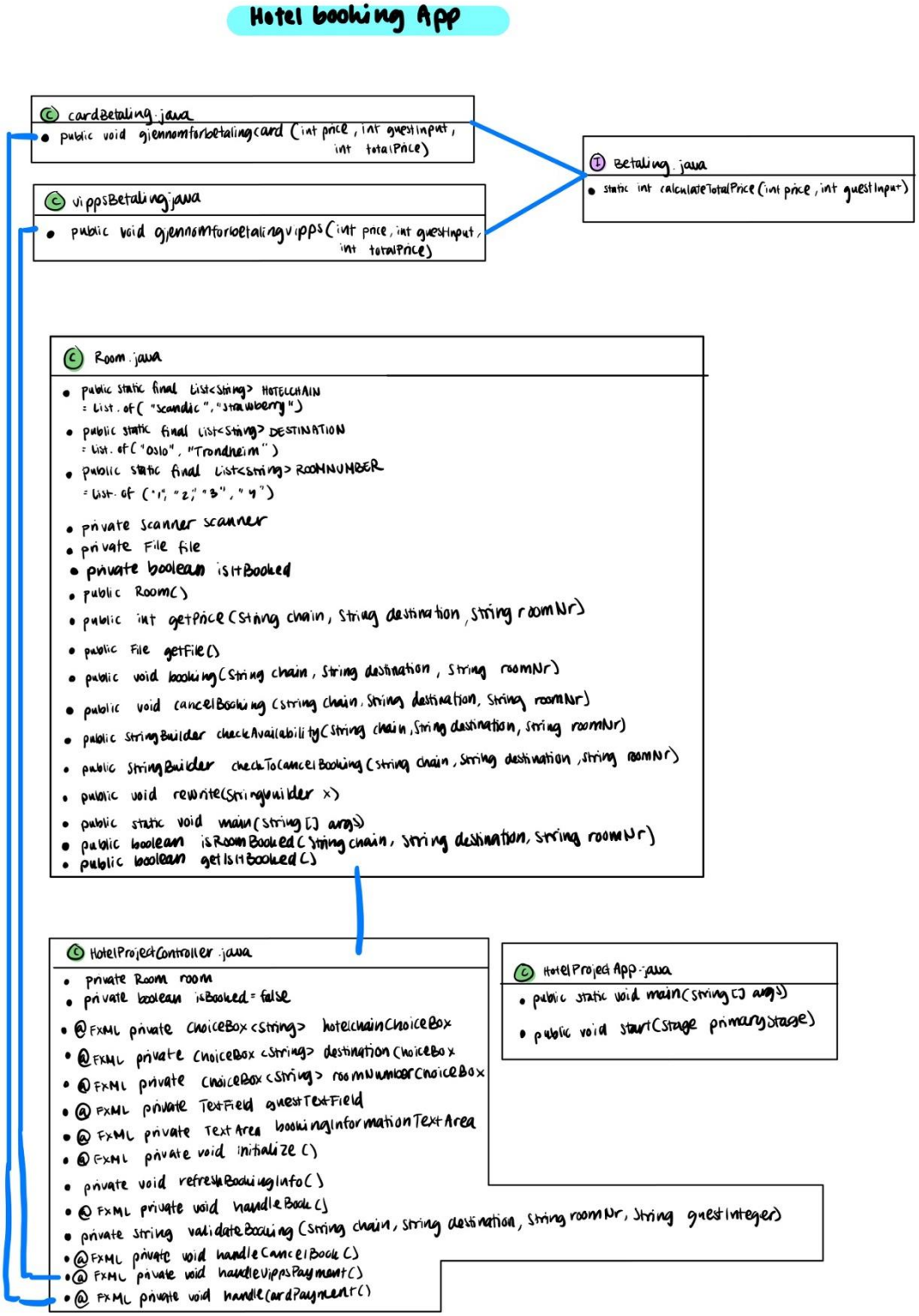
1. Beskrivelse av appen

Appen vår HotelProjectApp er et hotell booking app der brukeren kan velge hotellkjede, sted, hotellrom og angi antall gjester for så å booke rommet. Valgene til brukeren vil så bli lagret som informasjon. I tillegg har appen funksjonalitet i form av kalkulasjoner, ved å ta inn prisen per rom og antall gjester som variabler og gange dem sammen, blir total prisen til overnatting regnet ut og brukt videre når brukeren får valget med å betale med Vipps eller kort.

Når brukeren prøver å booke rommet, vil programmet lese fra en fil/scanne gjennom en csv fil der alle rommene er listet opp, programmet vil da sjekke om rommet er tilgjengelig, altså hvilke som ikke er booket fra før. Dersom rommet er tilgjengelig, vil brukeren kunne booke det og csv filen vil da endres til at rommet blir utilgjengelig. Dermed vil man ikke kunne dobbeltbooke rommet. Appen gir også mulighet til å kansellere bookingen, og vil da lignende endre tilstanden til rommet i csv filen.

Ved feil input som å skrive negative tall eller ikke tall når man skal angi antall gjester, ingen input, dobbelt booking av rom, dobbelt kansellering og dobbelt betaling vil appen gi error-popups med feilmelding. I likhet vil appen gi brukeren bekreftelse når hen booker, betaler og kansellerer en booking.

2. Diagram


Hotel booking App
cardBetaling.java
public void gjennomforbetalingcard (int price, int guestInput, int totalPrice)
vippsBetaling.java
public void gjennomforbetalingvipps (int price, int guestInput, int totalPrice)
Betaling.java
static int calculateTotalPrice (int price, int guestInput)
Room.java
public static final List<String> HOTELCHAIN = List.of("Scandic", "Strawberry")
public static final List<String> DESTINATION = List.of("Oslo", "Trondheim")
public static final List<String> ROOMNUMBER = List.of("1", "2", "3", "4")
private Scanner scanner
private File file
private boolean isItBooked
public Room()
public int getPrice(String chain, String destination, String roomNr)
public File getFile()
public void booking(String chain, String destination, String roomNr)
public void cancelBooking(String chain, String destination, String roomNr)
public StringBuilder checkAvailability(String chain, String destination, String roomNr)
public StringBuilder checkToCancelBooking(String chain, String destination, String roomNr)
public void rewrite(StringBuilder x)
public static void main(String[] args)
public boolean isRoomBooked(String chain, String destination, String roomNr)
public boolean getIsItBooked()
HotelProjectController.java
private Room room
private boolean isBooked = false
@FXML private ChoiceBox<String> hotelchainChoiceBox
@FXML private ChoiceBox<String> destinationChoiceBox
@FXML private ChoiceBox<String> roomNumberChoiceBox
@FXML private TextField guestTextField
@FXML private TextArea bookingInformationTextArea
@FXML private void initialize()
private void refreshBookingInfo()
@FXML private void handleBook()
private String validateBooking(String chain, String destination, String roomNr, String guestInteger)
@FXML private void handleCancelBook()
@FXML private void handleVippsPayment()
@FXML private void handleCardPayment()
HotelProjectApp.java
public static void main(String[] args)
public void start(Stage primaryStage)

3. Spørsmål

1. **Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For eksempel bruk av arv, interface, delegering osv.)**

Prosjekter dekker deler av pensumet som blir avholdt i emnet TDT4100, følgende pensum er klasser og objekter, innkapsling, tilstand og validering, interface, filhåndtering, enhetstesting og FXML. Klasser og objekter blir implementert gjennom de ulike klassene, eks. Room.java, vippsBetaling.java, cardBetaling.java, & Betaling.java osv. I Klassen room.java forekommer det en innkapsling ved å benytte synlighetsmodifikatorer, eks. public og private på enkelte metoder. Dette sikrer oss at metoder kan kun benyttes for Room.java, og ikke utenfor denne klassen. En annen innkapsling som blir benyttet i Room.java er å benytte en konstruktør for klassen, her blir tilstander som skal benyttes kun for objektet Room.java. Validering som unntaksfeil, f.eks. IllegalArgumentException og IllegalStateException blir også benyttes hos Room.java. Et eksempel på validering ligger i metoden booking(). Filhåndtering blir dekket av room.java, hvor metodene scanner og leser av en csv fil for så å endre på verdiene basert på brukerens handlinger. Interface blir dekket gjennom klassen betaling.java, her blir klassen implementert av cardBetaling.java og vippsBetaling.java. Enhetstesting blir dekket gjennom RoomTest.java, som sjekker at koden fungerer slik som den skal, samt teste input-verdier og tilstander som kan føre til feil. RoomTest.java tester også at koden fungerer slik som den skal. SceneBuilder ble benyttet for å designe appen, og gjennom dette programmet blir FXML skrevet og lest til vscode. Ved å ha en controller.java og skrive inn bestemte metoder som skal utføres når et spesifikt knapp trykkes, knyttes dette opp ved hjelp av FXML.

2. **Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?**

Deler av pensumet avholdt i emnet TDT4100 som ikke ble brukt i prosjektet kan være: arv og abstrakte klasser, comparator/comparable, iterator/iterable, deligering og observatører. For å forbedre eller gjøre appen mer komplett kan man i teorien implementere disse delene av pensum også i vårt prosjekt. En måte man kan integrere arv og abstrakte klasser inn i appen kan være hensiktsfullt dersom det fantes ulike typer rom som man kan booke i appen. Dette kunne da vært ulike familierom, enkeltrom, dobbeltrom osv. Her kunne man da hatt en abstrakt klasse for rom og flere underklasser til de ulike typene som arver egenskaper eller oppførsel fra den abstrakte klassen. I tillegg ble det ikke tatt i bruk grensesnittene Comparable/Comparator/Iterator/Iterable. Vi kunne tatt i bruk Comparable/Comparator hvis ville sammenligne rom basert på ulike kriterier som rom nummer eller pris. Vi kunne tatt i bruk Iterator/Iterable hvis vi ville implementere muligheten til å vise hvilke rom som var utilgjengelige/tilgjengelige ved å iterere gjennom samlingen av rom. Til slutt kunne man implementert delegering og observatører i prosjektet. Ved bruk av delegering kan man forenkle komplekse og store operasjoner ved å delegere ansvaret til andre klasser. Dette kunne for eksempel blitt gjort med room.java klassen som nå håndterer mange metoder og klasser. Ved delegering kunne vi lagd en egen klasse som bare håndterer booking bestillinger i stedet for å ha denne logikken i room.java klassen. Måten man kan integrere observatører i prosjektet, kan være å lage en observatør klasse som holder styr på og registrerer alle endringene som skjer på bookingstatusen eller betalingsstatusen.

3. **Hvordan forholder koden deres seg til Model-View-Controller-prinsippet? (Merk: det er ikke nødvendig at koden er helt perfekt i forhold til Model-View-Controller standarder. Det er mulig (og bra) å reflektere rundt svakheter i egen kode)**

Vi mener koden vår har et fint forhold til Model-View-Controller prinsippet, den er ikke helt perfekt og kan tilsynelatende endres litt på ved ekstra tid på prosjektet. Brukeren vil kunne se

appen (view), og view er definert av JavaFX elementene i Scenebuilder. Vi har prøvd vår beste å forenkle appen og gjøre ting opplagt med tanke på det vi har lært i et parallelt emne TDT4180 – Menneskemaskin interaksjon. Utover dette vil brukeren respondere til appen, og responsen som å klikke på en knapp vil aktivere kontrolleren. Det har blitt prøvd å gjøre kontrolleren enkelt som mulig, ved å ikke sette så mye logikk. I kontrolleren befinner det seg alert, og ulike catch handlinger ved error feil. En liten svakhet ved kontrolleren er at det eksisterer noen metoder der som ikke blir benyttet i FXML, men som er lurt å ha ettersom det krever input fra brukeren. Men dette er en essensiell del av appen med tanke på at korrekt innkapsling og validering for appen videre. Kontrolleren sender informasjon videre til modellen. I modellen ligger hovedmetodene våre i ulike klasser, innkapslet i ulike valideringer og feilhåndteringer. Informasjonen blir deretter sendt videre tilbake til kontrolleren for så å sende data til view. Brukeren får dermed muligheten til å handle videre med appen.

4. **Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke, hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at dere skal reflektere rundt egen bruk av tester)**

Bakgrunnen til valg av tester til prosjektet er at fokuset var på å teste den viktigste funksjonaliteten i appen da dette er vesentlig for at brukeren får en positiv opplevelse med appen i tillegg til muligheten til å bruke den slik den er tiltenkt å fungere. Dette blir da å kjøre selve appen, muligheten til å booke, avbestille, og betale. I tillegg til dette har vi valgt å teste feilhåndteringen: at det ikke skal være mulig å booke et utilgjengelig rom eller booke det to ganger, at det ikke skal være mulig å avbestille et allerede avbestilt rom, at det ikke skal være mulig å betale flere ganger enten med kort eller Vipps og at brukeren må skrive inn riktige verdier og ikke la være å skrive inn verdier når hen booker rom. Disse funksjonalitetene har vi valgt å teste fordi de er viktige for at brukeren skal oppleve appen som god, og at appen ikke krasjer eller oppfører seg uventet når det oppstår feil.