STUDENTOPPGAVE

NTNU

KANDIDATNUMMER(E):
997501 HÅKON LONGVA HARAM
997504 ERLEND NERÅS KNUDSEN
997507 ROBIN STAMNES THORHOLM
997521 BJØRNAR MAGNUS TENNFJORD

DATO:	FAGKODE:	STUDIUM:	ANT SIDER/BILAG:	BIBL. NR:
10.03.17	ID101805	Automatiseringsteknikk	6/0	

VEILEDER(E) :

Arne Styve

TITTEL :

Obligatorisk Innlevering 3 (Fase 2)

SAMMENDRAG:

Rapporten inneholder første del av gruppeoppgaven i faget ID101912 Objektorientert Programmering. Vi har beskrevet hvordan programmet fungerer hittil og hvilke løsninger vi har valgt for de ulike klassene.

Denne oppgaven er en eksamensbesvarelse utført av studenter ved NTNU i Ålesund.

INNHOLD

1 SAMMENDRAG

Rapporten inneholder første del av gruppeoppgaven i faget ID101912 Objektorientert Programmering. Vi har beskrevet hvordan programmet fungerer hittil og hvilke løsninger vi har valgt for de ulike klassene.

2 TERMINOLOGI

IDE - Integrated Development Environment

3 INNLEDNING – PROBLEMSTILLING

I dette programmet skal vi skape en booking løsning for et flyselskap. Dette programmet skal håndtere salg av flybilletter.

I innlevering 2 (fase 1) skulle programmet bestå av klassene:

- Passenger
- Seat
- SeatRegister

I innlevering 3 (fase 2) skulle programmet bestå av klassene:

- Flight
- FlightRegister
- PassengerRegister
- Ticket
- TicketRegister
- Application
- ApplicationUI
- Junit test
    - FlightTest
    - PassengerTest
    - SeatTest
    - TicketTest

Disse klassene skal være så modulerbare som mulig slik vi kan bruke de i de kommende innleveringene uten å skrive om kodene til klassene.

3.1 Bakgrunn

3.2 Formål og problemstilling

3.3 Avgrensninger

3.4 Rapportens oppbygning

4 BAKGRUNN - TEORETISK GRUNNLAG

I dette programmet bruker vi følgende klasser:

- **Passenger** skal inneholde fornavn, etternavn og epost adresse. Denne generelle klassen oppretter passasjerer som objekter, med informasjon fra brukeren som legger inn personene.
- **Seat** skal inneholde sete-rad, setebokstav (kolonne) og status for tilgjengelighet. Den generelle klassen oppretter seter som objekter med informasjon gitt av en ekstern input.
- **SeatRegister** generer et seteregister med et gitt antall rekker og kolonner av klassen Seat. Der du har muligheten til å legge til, fjerne, søke etter seter, samt hente ut antall eller en liste over tilgjengelige/utilgjengelige seter.
- **Flight** holder informasjon om flyID, destinasjon, avgangssted, avgangstid, ankomsttid, avgangsdato og ankomstdato. Man har mulighet til å endre disse verdiene for utenom flightID, destinationAirport og departureAirport siden disse verdiene aldri vil endres.
- **Ticket** holder på informasjonen som står på billetten til passasjeren.

Vi sørger for at alle på gruppen følger den samme gode kodestilen. Dette består blant annet av at for løkker ikke skal brytes, hver klasse har et ansvarsområde, hver metode skal kun ha en oppgave og felt i klasser skal være private.

5 METODE – DESIGN

Til innlevering 2 (fase 1) har vi brukt BlueJ of NetBeans som IDE. Vi brukte versjonskontrollsystemet Git og Git klienten SourceTree på slutten av innleveringen når programmet fungerte etter kravspesifikasjonen. På denne måten har vi en versjonskontroll på den første innlevering og som er klar til bruk på innlevering 3 (fase 2).

Gjennom hele utviklingen sørger vi for at de generelle klassene er uavhengig av hverandre, dette gjør løsningen modulerbar og mer mottakelig for senere utvidelser og oppdateringer.

Til innlevering 3 (fase 2) brukte vi helt og holdent NetBeans som IDE. Vi fortsatte bruken av Git men nå også for å synkronisere arbeidet vårt

6 BEARBEIDING OG RESULTATER

I følge kravspesifikasjonen så skulle vi identifisere de nødvendige klassene for et system som skulle kunne opprette et sete, en person og et register av setene.

I sete-klassen har vi implementert metoder for å opprette et sete, hente ut sete-IDen, sjekke om setet er ledig, og sette statusen til tilgjengelig/utilgjengelig.

Passasjer-klassen gir brukeren muligheten til å hente ut navnet til den aktuelle passasjeren, emailen, samt skrive ut alle detaljene om passasjeren.

Seteregisteret er den mest avanserte klassen. Her holder klassen styr på en liste av seter i en ArrayList av typen Seat. I konstruktøren angir man ønsket antall rader og kolonner med seter. Her har vi laget en egen metode (fillSeats) som fyller opp listen av seter med objekter av ønsket antall rader og kolonner. Her har vi implementert metoder for å kunne legge til, fjerne og søke etter seter. En kan også hente ut selve objektet av et sete i arraylisten, liste opp alle seter/tilgjengelige/utilgjengelige seter, i tillegg til antall seter.

Til fase 2 av prosjektet laget vi en enkelt UI system som presenterer funksjonaliteten på programmet til brukeren på en enkel og oversiktlig måte. Vi har også fjernet system out print i de underliggende klassene av application UI. På denne måte så er det kun applicationUI som behandler interaksjonen mellom programmet og brukeren.

7 DRØFTING OG KONKLUSJON

En av de store utfordringene i denne oppgaven var å legge en plan om hvordan vi skulle bygge opp programmet og gjøre det klart for senere innleveringer.

Vi har erfart at å lage klassene uavhengig av hverandre er veldig nyttig for å få en bedre oversikt over programmet. At en og en klasse er spesialisert til å utføre en spesiell oppgave gjør programmet mer modulerbart.

Selv om det var mye å gjøre, så fikk vi gjort det som skulle gjøre i henhold til innlevering 2.

De viktigste og mest pålitelige kildene våre var boken «Objects First with Java» og Java Class Library dokumentet på nettet. De ulike forumene på nettet har meget ulike løsninger og kodestiler, dette kunne raskt føre til mer forvirring enn hjelp så dette ble mer brukt til inspirasjon enn et oppslagsverk.

Vi fikk implementert de utvidelsene som var beskrevet i innlevering 3 (fase 2). Vi har ryddet opp i en god del av koden og gjort

8 KONKLUSJON – ERFARING

Se punkt 7.

9 REFERANSER

[1] ”Objects First With Java”, Sixth edition, av Barnes og Kölling. ISBN ….

[2] http://docs.oracle.com/javase/8/docs/api/

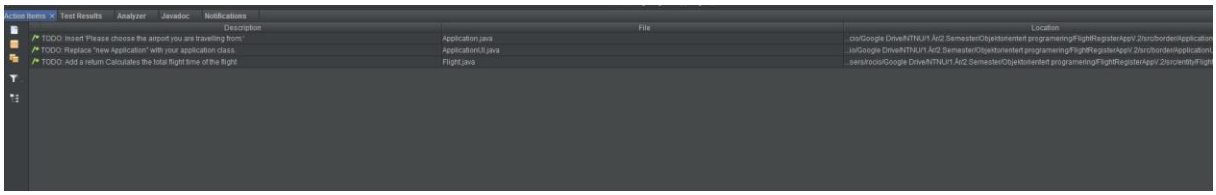
10 VEDLEGG

FlightRegisterAppV.2 (Mappe)

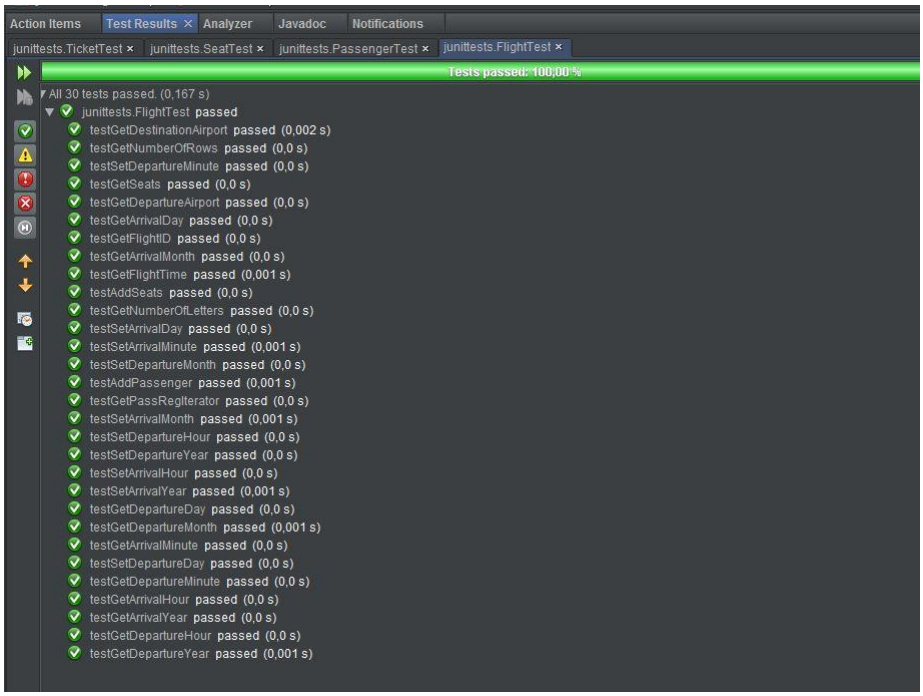
10.1 Javadoc:

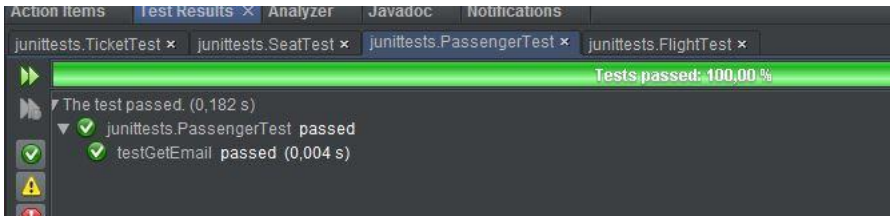
\FlightRegisterAppV.2\dist\javadoc

10.2 Checkstyle:

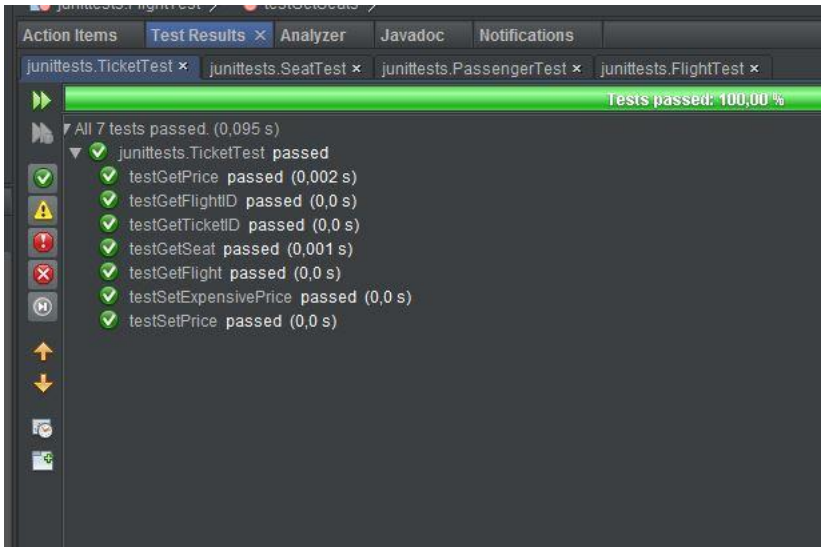


JUnit test:






junittests.TicketTest ×
junittests.SeatTest ×
junittests.PassengerTest ×
junittests.FlightTest ×
Tests passed: 100,00 %
All 5 tests passed. (0,136 s)
junittests.SeatTest passed
testIsAvailable passed (0,002 s)
testSetUnavailable passed (0,0 s)
testGetSeatId passed (0,001 s)
testSetAvailable passed (0,005 s)
testSeatNumberForNegatives passed (0,001 s)


Action Items
Test Results
Analyzer
Javadoc
Notifications
junittests.TicketTest ×
junittests.SeatTest ×
junittests.PassengerTest ×
junittests.FlightTest ×
Tests passed: 100,00 %
All 7 tests passed. (0,095 s)
junittests.TicketTest passed
testGetPrice passed (0,002 s)
testGetFlightID passed (0,0 s)
testGetTicketID passed (0,0 s)
testGetSeat passed (0,001 s)
testGetFlight passed (0,0 s)
testSetExpensivePrice passed (0,0 s)
testSetPrice passed (0,0 s)