

Mappeoppgave

IDATx1003 Programmering 1 - Del 1

Høst 2023

Innhold

Dette dokumentet inneholder 3 deler:

1. Om mappevurdering - her finner du en detaljert beskrivelse av hvordan mappevurdering gjennomføres i emnet, hvilke deler mappen består av, og hvordan mappen skal leveres
2. Beskrivelse av prosjektet i sin helhet (kravspek)
3. Hva som skal gjøres i del 1 av 3 - **NB! LES DENNE GRUNDIG FØR DU STARTER MED KODINGEN!!**

Om mappevurdering i IDATx1003

I dette emnet er det ingen eksamen (skriftlig eller muntlig). I stedet for benytter vi **Mappe** som vurderingsform.

Mappen leveres ut ca uke 40 og går parallelt med øvrige øvingsaktiviteter ut semesteret.

Detaljert prosjektbeskrivelse blir publisert i 3 deler:

- **Del 1 - ca Uke 40:** Omhandler design av klasse, kodelstil og dokumentasjon samt versjonskontroll og enhetstesting.
- **Del 2 - ca Uke 43:** Omhandler samling av objekter (ArrayList, HashMap osv), håndtering av samlinger og søk i samlinger (Lambda-uttrykk og streams/filter)
- **Del 3 - ca Uke 45:** Omhandler ferdigstilling av applikasjon med et tekstbasert brukergrensesnitt med komplette enhetstester.

Hver ny del bygger på foregående del. Når ny del publiseres, vil du få mulighet til **muntlig tilbakemelding** på arbeidet du har gjort så langt, med mulighet å justere/endre/forbedre din løsning.

Du vil få i alt 3 slike tilbakemeldinger, der den siste tilbakemeldingen gis i uke 47.

Endelig **fungerende** løsning/applikasjon leveres i GitHub (Ålesund & Gjøvik)/GitLab (Trondheim) samt som ZIP-fil **sammen med en rapport i Inspira** innen kl 14:00 tirsdag 13/12. Inspira åpnes opp for innlevering mandag 12/12 kl 09:00. (Se Blackboard for detaljer om GitHub/GitLab)

Vekting mellom rapport og prosjekt mot endelig karakter:

- Rapport - 30%
- Prosjekt - 70%

Kilder:

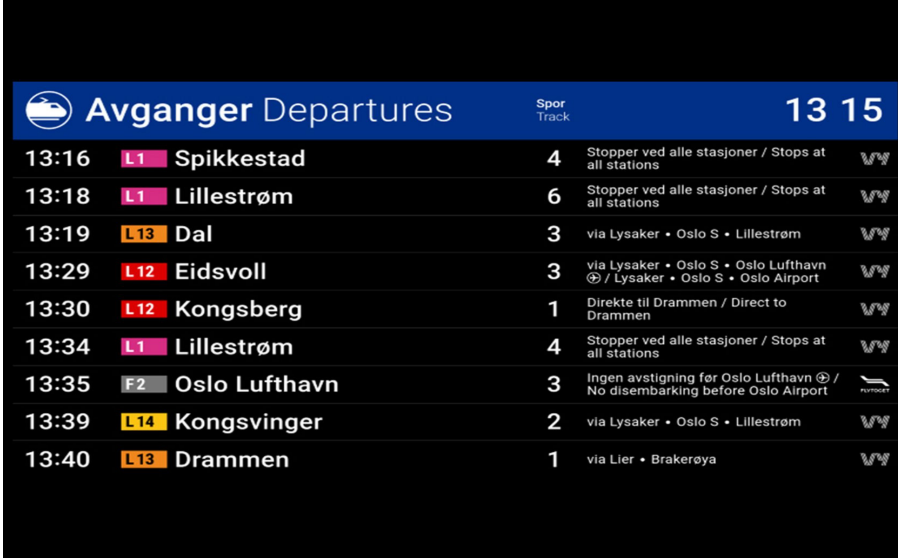
- Hva Forskriften sier om mappe som vurderingsform:
https://lovdata.no/dokument/SF/forskrift/2015-12-08-1449/KAPITTEL_5#%C2%A75-11

Prosjektoppgaven - Train Dispatch System

Innledning

I dette prosjektet skal du utvikle et forenklet system for avvikling av **togavganger** (engelsk: "Train Dispatch System").

Under ser du eksempel på en typisk oversikt over **togavganger** fra en stasjon.



Avganger Departures		Spor Track	13 15
13:16	L1 Spikkestad	4	Stopper ved alle stasjoner / Stops at all stations
13:18	L1 Lillestrøm	6	Stopper ved alle stasjoner / Stops at all stations
13:19	L13 Dal	3	via Lysaker • Oslo S • Lillestrøm
13:29	L12 Eidsvoll	3	via Lysaker • Oslo S • Oslo Lufthavn ⊕ / Lysaker • Oslo S • Oslo Airport
13:30	L12 Kongsberg	1	Direkte til Drammen / Direct to Drammen
13:34	L1 Lillestrøm	4	Stopper ved alle stasjoner / Stops at all stations
13:35	F2 Oslo Lufthavn	3	Ingen avstigning før Oslo Lufthavn ⊕ / No disembarking before Oslo Airport
13:39	L14 Kongsvinger	2	via Lysaker • Oslo S • Lillestrøm
13:40	L13 Drammen	1	via Lier • Brakerøya

Figur 1 Toginfotavle på en stasjon

(Kilde: <https://www.banenor.no/apne-data-fa-togtidene-gratis-pa-nett-og-skjermer/>)

En **togavgang** (engelsk: "Train departure") består som regel av følgende informasjon:

- *Avgangstid* (engelsk: "departure time") - klokkeslettet for når toget går, på formatet tt:mm med 24 timers visning
- *Linje* (engelsk: "line") - en tekst å formen "L1", "F4" osv. Definerer en strekning som toget kjører på. Flere ulike tog kan kjøre samme strekning da til ulike tidspunkt.
- *Tognummer* (engelsk: "train number") - en tekst med **et unikt nummer** innenfor samme dag på formen 602, 45, 1951 etc. Nummeret er unikt innenfor et 24-timers vindu
- *Destinasjon* (engelsk: "destination")
- *Spor* (engelsk: "track") - Et heltall som angir hvilket spor toget er satt opp på. Dersom toget ikke har fått tildelt spor ennå, settes spor til -1.

- *Forsinkelse* (engelsk: "delay") - Angir hvor mye toget er forsinket i timer og minutter. Hvis ingen forsinkelse, settes 00:00.

For eksempler på ulike **linjer** og **tognummer**, se rutetidene til VY: <https://www.vy.no/trafikk-og-ruter/rutetider?item=2893>

Tog nr	62	602	64	66	66	66	606
07.08.2023-09.12.2023							
Periode				1.10-8.12.	1.5-30.9.	K =	
Mandag-Fredag	M-F	M-F	M-F	F	M-F		M-F
Lørdag	L	L	L		L		
Søndag	S	S	S	S	S		S

Figur 2 Sammenheng mellom "linje" og "tognummer"

I Figur 2 ser vi eksempel fra en rutetabell som viser **linje** F4 med alle avganger, og **tognummer** til hvert av togavgangene (62, 602, 64 osv).

Avgrensninger i oppgaven

Systemet som skal utvikles har følgende avgrensninger:

- Systemet skal støtte **kun en stasjon** (altså tog som kjører fra én bestemt stasjon).
- Systemet tar ikke hensyn til **dato**, kun **tidspunkt innenfor en dag**.
- "**Klokken**" oppdateres manuelt fra brukermenyen (ingen bruk av systemklokke osv).

Funksjonelle krav

Applikasjonen som skal utvikles, skal ha et **tekstbasert brukergrensesnitt** i form av en **meny**.

Fra denne menyen, skal operatøren på stasjonen kunne gjøre følgende oppgaver (ikke begrenset til):

- Vise/skrive ut oversikt over togavgangene, sortert etter avreisetidspunkt (informasjonstavle).
- Legge inn en ny togavgang – det skal ikke være mulig å legge inn et tog med tognummer tilsvarende eksisterende tog i listen.
- Tildele spor til en togavgang – ved først å søke opp togavgang basert på tognummer, og så sette spor.
- Legg inn forsinkelse på en togavgang – ved å først søke etter en gitt togavgang basert på tognummer, og deretter legge til forsinkelse.
- Søke etter en togavgang basert på Tognummer
- Søke etter togavgang basert på destinasjon
- Oppdatere klokken (tidspunktet på dagen) – ved å spørre bruker etter nytt klokkeslett.
- Avslutte applikasjonen

En togavgang skal automatisk fjernes fra oversikten dersom avreisetidspunktet (pluss eventuell forsinkelse) er tidligere enn klokken (tidspunktet på dagen). Eksempel: har et tog avreisetidspunkt kl 12:35 og er 5 minutter forsinket, skal togavgangen vises i oversikten så lenge klokken (tidspunktet på dagen) er tidligere enn 12:40. I det øyeblikket klokken passerer 12:40, skal togavgangen fjernes fra oversikten.

Når klokken settes til nytt klokkeslett, skal det ikke være mulig å sette tidspunktet tidligere enn gjeldende tidspunkt. M.a.o. er klokken 12:45, skal det ikke være mulig å sette klokken til 12:40.

Informasjonstavlen

Du velger selv hvordan du vil presentere din **informasjonstavle** (tilsvarende den i Figur 1), men tavlen **skal** vise følgende informasjon (i følgende rekkefølge):

- Avgangstid på formatet "hh:mm"
 - Linje
 - Tognummer
 - Destinasjon
 - Eventuell forsinkelse. Hvis ingen forsinkelse skal det ikke vises noe (heller ikke "00:00")
 - Spor. Hvis ingen spor tildelt, skal ingen ting vises.
-

Mappen Del 1 (av 3)

I første del av mappen skal du fokusere på å:

- forstå oppgaven/prosjektet
- få opprettet prosjektet som et Maven-prosjekt
- sette prosjektet under versjonskontroll koblet til GitHub/GitLab
- implementere **entitetsklassen** (klassen som representerer en togavgang)
- implementere **testklasse** for å teste entitetsklassen.
- starte på rapporten

Detaljene følger under.

Kodeprosjekt

Du får tilgang til et ferdig oppsatt prosjekt via **GitHub Classroom** for mappen. For å få tilgang til dette prosjektet og få opprettet ditt eget repository for Mappen på GitHub, se BlackBoard og siden "[Opprette Mappe-prosjektet fra GitHub](#)" under menyen "Mappevurdering" i Black Board.

Når du har fått opprettet prosjektet og **klonet** prosjektet til din egen datamaskin, åpner du prosjektet i din IDE (IntelliJ, VSCode el.l.), og følger videre instruksjer under:

- Lage **entitetsklasse** for tog-avgang (engelsk: "Train Departure")
- Dokumenter klassen grundig, inkludert
 - Rolle/ansvar
 - Hvilke informasjon klassen holder på og hvilken datatyper du har valgt for hver info og hvorfor (begrunnelse)
 - En vurdering av hvilke informasjon skal kun settes ved opprettelse av instans (m.a.o. ingen set-metoder for disse), og hvilke informasjon må kunne endres etter at instansen er opprettet
 - Hvordan klassen responderer på ugyldige data - hvilken strategi følger klassen (kaster unntak? setter dummy-verdi?)
- Implementer nødvendige felt i klassen, med egnede **datatyper**. Tips: for **tid** kan det være lurt å bruke klassen [LocalTime](#)
- Lag **enhetstester** for å teste at klassen for togavgang fungerer som den skal og er robust. Husk både *positive*- og *negative* tester.
- Kjør **CheckStyle** på koden din med **Google-reglene**.
- Opprett en klasse som på sikt skal bli brukergrensesnittet og dermed ta seg av all brukerinteraksjon. Opprett to metoder; **start()** og **init()**.
- Bruk start-metoden til å skrive **enkel testkode** for å teste at Train-Departure-klassen fungerer iht spek ved f.eks. å opprette 3-4 instanser av klassen, og skrive ut objektene (hint: Kan lønne seg å lage en egen metode for å skrive ut detaljene til en togavgang...).
- Opprett til slutt en klasse som skal være selve applikasjonen og som inneholder **public static void main(String[] args)**-metoden.
Opprett main()-metoden. La denne metoden lage en instans av UI-klassen og kall deretter metodene **init()** og **start()** til UI-objektet.

Husk å utfør **commit (innsjekk)** til lokalt Git-repository jevnlig/hver gang du gjør endringer i koden. Husk å alltid legge til en kommentar til innsjekken.

Endelig prosjekt skal pushes til GitHub/GitLab.

Rapport

Begynn på rapporten allerede nå:

- Sett deg inn i rapportmalen
- Fyll ut forside, og skriv innledningen
- Lag et **UseCase-diagram** som viser hvilken funksjonalitet løsningen skal tilby en bruker.
- Start på kapitlene om **teori** og **metode**.
- I **resultat**-kapitlet kan du allerede nå gi en kort beskrivelse av hvilke klasser du ser for deg du vil lage i prosjektet, og vise et klassediagram som viser avhengigheten mellom de.

Viktige sjekkpunkter

Når du løser oppgaven, bør du dobbeltsjekke følgende:

- Kompilering/bygging og prosjektstruktur:
 - Er prosjektet et Maven-prosjekt med en ryddig og riktig katalogstruktur?
 - Kan prosjektet **kompile** uten feil (*mvn compile*)?
 - Er prosjektet plassert på fornuftig sted på din harddisk (finner du lett tilbake til prosjektet)?
- Versjonskontroll med git:
 - Er prosjektet underlagt versjonskontroll med sentralt repository i GitHub eller GitLab?
 - Finnes det flere innsjekkinger (commits) ?
 - Beskriver commit-meldingene endringene på en kort og konsis måte?
- Enhetstester:
 - Har enhetstestene beskrivende navn som dokumenterer hva testene gjør?
 - Følger de mønstret Arrange-Act-Assert?
 - Tas det hensyn til både positive og negative tilfeller?
 - Er testdekningen god nok?
 - Kjører samtlige tester uten feil?
- Er klassene for togavgang, brukergrensesnitt og applikasjon implementert iht oppgavebeskrivelsen?
- Kodekvalitet:
 - Er koden godt dokumentert iht JavaDoc-standard?
 - Er koden robust (validering mm)?
 - Har variabler, metoder og klasser beskrivende navn?
 - Er klassene gruppert i en logisk pakkestruktur?